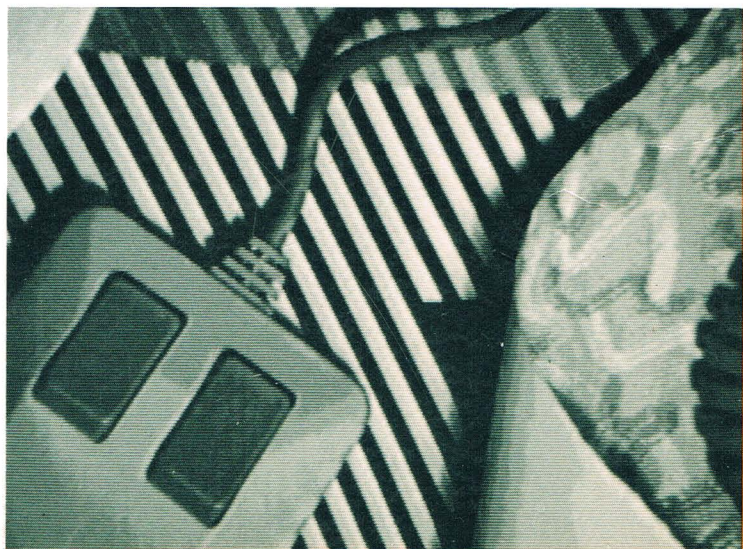


AMIGA™

ROM Kernel

Reference Manual:
Libraries and Devices

Commodore Business Machines, Inc.



Amiga
ROM Kernel Reference Manual
Libraries and Devices

Commodore Business Machines, Inc.

Amiga Technical Reference Series



Addison-Wesley Publishing Company, Inc.

Reading, Massachusetts Menlo Park, California Don Mills, Ontario
Wokingham, England Amsterdam Sydney Singapore Tokyo
Mexico City Bogota Santiago San Juan

Written by Rob Peck

Contributing editors:

Dave Berezowski, Bob Burns, Susan Deyl, Sam Dicker, Andy Finkel, Larry Hildenbrand, Neil Katin, Dale Luck, and R. J. Mical

Program examples by Rob Peck, Sam Dicker, Tom Pohorsky, Larry Hildenbrand, and Neil Katin

The following people have contributed significantly to the contents of this manual:

Bruce Barrett, Dave Lucas, Jim Mackkraz, Bob Pariseau, Tom Pohorsky, Stan Shepard, and Barry Whitebook

This book is dedicated to all those "busy guys" who made Amiga and who are Amiga.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book and Addison-Wesley was aware of a trademark claim, the designations have been printed in initial caps.

COPYRIGHT © 1986 by Commodore Electronics, Ltd.

Library of Congress Cataloging-in-Publication Data

Amiga ROM kernel reference manual.

(Amiga technical reference series)

Includes index.

1. Amiga (Computer)--Programming. 2. Read-only storage. I. Commodore Business Machines. II. Series.
QA76.8.A177A6554 1986 005.4'46 86-10878
ISBN 0-201-11078-4

CDEFGHIJ-AL-89876

Third Printing, September 1986

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Printed in the United States of America. Published simultaneously in Canada.

DISCLAIMER

COMMODORE-AMIGA, INC., ("COMMODORE") MAKES NO WARRANTIES, EITHER EXPRESSED OR IMPLIED, WITH RESPECT TO THE PROGRAMS DESCRIBED HEREIN, THEIR QUALITY, PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. THESE PROGRAMS ARE SOLD "AS IS." THE ENTIRE RISK AS TO THEIR QUALITY AND PERFORMANCE IS WITH THE BUYER. SHOULD THE PROGRAMS PROVE DEFECTIVE FOLLOWING PURCHASE, THE BUYER (AND NOT THE CREATOR OF THE PROGRAMS, COMMODORE, THEIR DISTRIBUTORS OR THEIR RETAILERS) ASSUMES THE ENTIRE COST OF ALL NECESSARY DAMAGES. IN NO EVENT WILL COMMODORE BE LIABLE FOR DIRECT, INDIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT IN THE PROGRAMS EVEN IF IT HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME LAWS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF IMPLIED WARRANTIES OR LIABILITIES FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY.

Amiga is a trademark of Commodore-Amiga, Inc.

Printed from camera-ready mechanicals supplied by the authors.

PREFACE

System Software Architecture

The Amiga kernel consists of a number of system modules, some of which reside permanently in the protected *kickstart* memory and others that are loaded as needed from the system disk. Figure P-1 illustrates how the various modules interact with one another. At the top of the hierarchy are Workbench and the Command Line Interface (CLI), the user-visible portions of the system. Workbench uses Intuition to produce its displays and AmigaDOS to interact with the filing system. Intuition, in turn, uses the input device to retrieve its input and the graphics and layers library routines to produce its output.

AmigaDOS controls processes and maintains the filing system and is in turn built on Exec, which manages tasks, task switching, interrupt scheduling, message-passing, I/O, and many other functions.

At the lowest level of the hierarchy is the Amiga hardware itself. Just above the hardware are the modules that control the hardware directly. Exec controls the 68000, scheduling its time among tasks and maintaining its interrupt vectors, among other things. The trackdisk device is the lowest-level interface to the disk hardware, performing disk-head movement and raw disk I/O. The keyboard and gameport devices handle the keyboard and gameport hardware, queuing up input events for the input device to

process. The audio device, serial device, and parallel device handle their respective hardware. Finally, the routines in the graphics library handle the interface to the graphics hardware.

Programming

The functions of the kernel were designed to be accessed from any language that follows the Amiga's standard interface conventions. These conventions define the proper naming of symbols, the correct usage of processor registers, and the format of public data structures.

REGISTER CONVENTIONS

All system functions follow a simple set of register conventions. The conventions apply when any system function is called; programmers are encouraged to use the same conventions in their own code.

The registers D0, D1, A0, and A1 are always scratch; they are free to be modified at any time. A function may use these registers without first saving their previous contents. The values of all other data and address registers must first be preserved. If any of these registers are used by a function, their contents must be saved and restored appropriately.

If assembly code is used, function parameters may be passed in registers. The conventions in the preceding paragraphs apply to this use of registers as well. Parameters passed in D0, D1, A0, or A1 may be destroyed. All other registers must be preserved.

If a function returns a result, it is passed back to the caller in D0. If a function returns more than one result, the primary result is returned in D0 and all other results are returned by accessing reference parameters.

The A6 register has a special use within the system, and it may not be used as a parameter to system functions. It is normally used as a pointer to the base of a function vector table. All kernel functions are accessed by jumping to an address relative to this base.

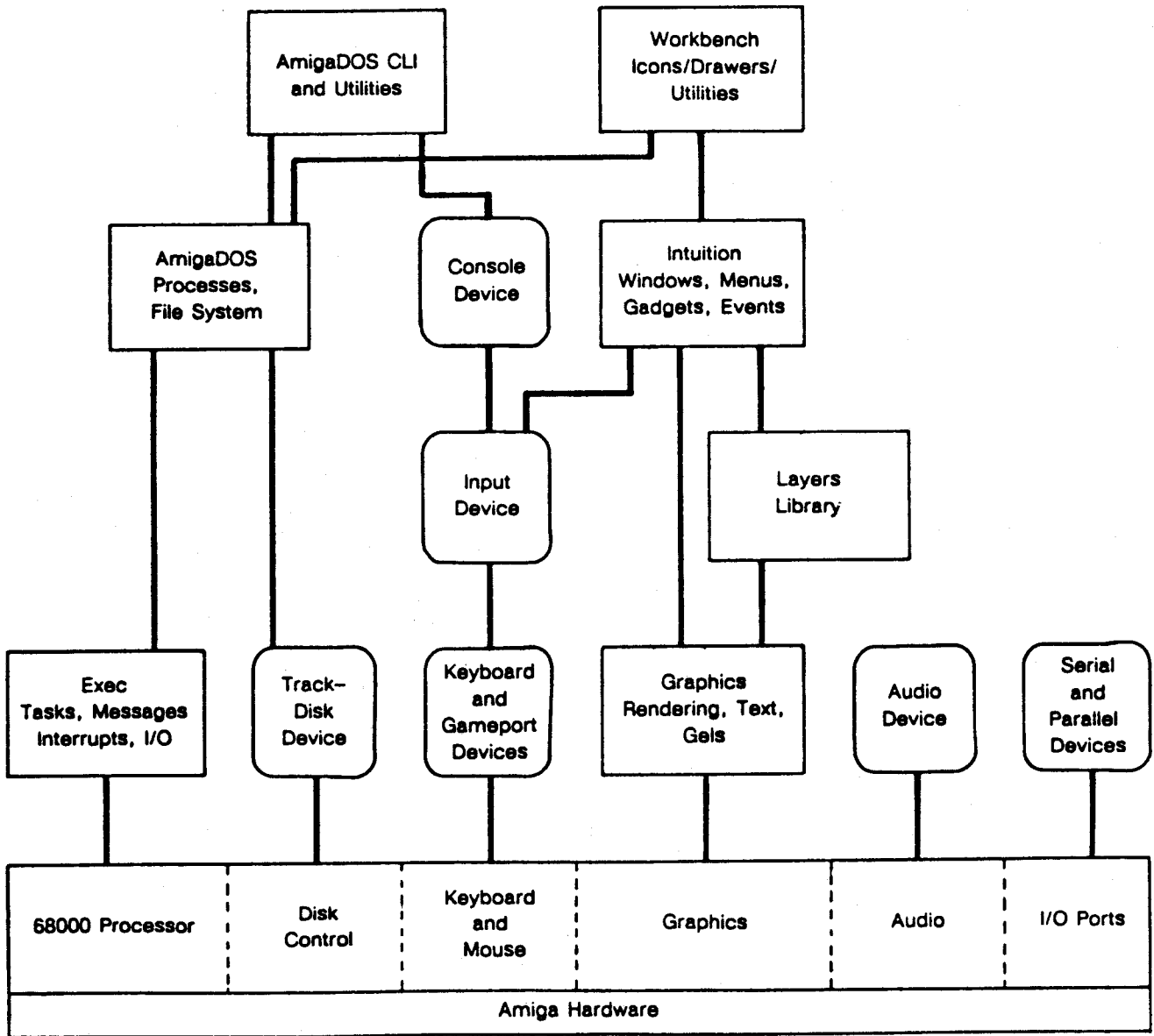


Figure P-1: Amiga System Software Modules

DATA STRUCTURES

The naming, format, and initial values of public data structures must also be consistent. The conventions are quite simple and are summarized below.

1. All non-byte fields must be word-aligned. This may require that certain fields be padded with an extra byte.
2. All address pointers should be 32 bits (not 24 bits) in size. The upper byte must never be used for data.
3. Fields that are not defined to contain particular initial values must be initialized to zero. This includes pointer fields.
4. All reserved fields must be initialized to zero (for future compatibility).
5. Data structures to be accessed by custom hardware must not be allocated on a program stack.
6. Public data structures (such as a task control structure) must not be allocated on a program stack.
7. When data structures are dynamically allocated, conventions 3 and 4 above can be satisfied by specifying that the structure is to be cleared upon allocation.

OTHER PRACTICES

A few other general programming practices should be noted.

1. Never use absolute addresses. All hardware registers and special addresses have symbolic names (see the include files and *amiga.lib*).
2. Because this is a multitasking system, programs must never directly modify the processor exception vectors (including traps) or the processor priority level.
3. Do not assume that programs can access hardware resources directly. Most hardware is controlled by system software that will not respond well to interference. Shared hardware requires programs to use the proper sharing protocols.
4. Do not access shared data structures directly without the proper mutual exclusion. Remember, it is a multitasking system and other tasks may also be accessing the same structures.

5. Most system functions require a particular execution environment. For example, DOS functions can be executed only from within a process; execution from within a task is not sufficient. As another example, most kernel functions can be executed from within tasks, but cannot be executed from within interrupts.
6. The system does not monitor the size of a program stack. Take care that your programs do not cause it to overflow.
7. Tasks always execute in the 68000 processor user mode. Supervisor mode is reserved for interrupts, traps, and task dispatching. Take extreme care if your code executes in supervisor mode. Exceptions while in supervisor mode are deadly.
8. Do not disable interrupts or multitasking for long periods of time.
9. Assembly code functions that return a result do not necessarily affect the processor condition codes. By convention, the caller must test the returned value before acting on a condition code. This is usually done with a **TST** or **MOVE** instruction. Do not trust the condition codes returned by system functions.

68010 AND 68020 COMPATIBILITY

If you wish your code to be upwardly compatible with the 68010/68020 processors, you must avoid certain instructions and you must not make assumptions about the format of the supervisor stack frame. In particular, the **MOVE SR,<ea>** instruction is a privileged instruction on the 68010 and 68020. If you want your code to work correctly on all 680x0 processors, you should use the **GetCC()** function instead (see the Exec library function descriptions in the "Library Summaries" appendix of this book).

Contents of This Manual

This manual describes the graphics support routines (including text and animation), the I/O devices, the Workbench (an environment for running programs), and the floating point mathematics library. For information about the multitasking executive, see *Amiga ROM Kernel Reference Manual: Exec*.

The discussion of the data structures and routines in this manual is reinforced through numerous C-language examples. The examples are kept as simple as possible. Whenever possible, each example demonstrates a single function. Where appropriate, there are complete sample programs.

Boldface type is used for the names of functions, data structures, macros, and variables. System header files and other system file names are shown in italics.

For more information about system software, see *Amiga Intuition Reference Manual*, *AmigaDOS User's Manual*, *AmigaDOS Developer's Manual*, and *AmigaDOS Technical Reference Manual*.

Contents

PART I

Chapter 1 Graphics Primitives	1
Introduction	2
COMPONENTS OF A DISPLAY	3
INTRODUCTION TO RASTER DISPLAYS	3
INTERLACED AND NON-INTERLACED MODES	5
HIGH- AND LOW-RESOLUTION MODES	6
FORMING AN IMAGE	6
ROLE OF THE COPPER (COPROCESSOR)	9
Display Routines and Structures	9
LIMITATIONS ON THE USE OF VIEWPORTS	11
CHARACTERISTICS OF A VIEWPORT	12
VIEWPORT SIZE SPECIFICATIONS	12
VIEWPORT COLOR SELECTION	13
VIEWPORT DISPLAY MEMORY	19
FORMING A BASIC DISPLAY	21
LOADING AND DISPLAYING THE VIEW	25
GRAPHICS EXAMPLE PROGRAM	26
Advanced Topics	30
CREATING A DUAL-PLAYFIELD DISPLAY	30
CREATING A DOUBLE-BUFFERED DISPLAY	32
HOLD-AND-MODIFY MODE	34
Drawing Routines	35
INITIALIZING A BITMAP STRUCTURE	35
INITIALIZING A RASTPORT STRUCTURE	35
USING THE GRAPHICS DRAWING ROUTINES	42
User Copper Lists	61
Advanced Graphics Examples	65
DUAL-PLAYFIELDS EXAMPLE	65
HOLD-AND-MODIFY MODE EXAMPLE	69
Chapter 2 Layers	77
Introduction	77
DEFINITION OF LAYERS	78
TYPES OF LAYERS SUPPORTED	79
Layers Library Routines	79
INITIALIZING AND DEALLOCATING LAYERS	80
INTERTASK OPERATIONS	81
CREATING AND DELETING LAYERS	82

MOVING LAYERS	82
SIZING LAYERS	82
CHANGING A VIEWPOINT	83
REORDERING LAYERS	83
DETERMINING LAYER POSITION	83
SUB-LAYER RECTANGLE OPERATIONS	84
The Layer's RastPort	84
SIMPLE REFRESH LAYER	85
SMART REFRESH LAYER	86
SUPERBITMAP LAYER	86
BACKDROP LAYER	86
Using the Layers Library	87
OPENING THE LAYERS LIBRARY	87
OPENING THE GRAPHICS LIBRARY	87
CREATING A VIEWING WORKSPACE	88
CREATING THE LAYERS	88
GETTING THE POINTERS TO THE RASTPORTS	89
USING THE RASTPORTS FOR DISPLAY	89
LAYERS EXAMPLE	89
Clipping Rectangle List	94
DAMAGE LIST	94
REPAIRING THE DAMAGE	94
Regions	95
CREATING AND DELETING REGIONS	95
CHANGING A REGION	96
CLEARING A REGION	96
USING REGIONS	96
SAMPLE APPLICATION FOR REGIONS	98
Chapter 3 Animation	103
Introduction	103
PREPARING TO USE GRAPHICS ANIMATION	104
TYPES OF ANIMATION	104
THE GELS SYSTEM	106
Using Simple (Hardware) Sprites	110
CONTROLLING SPRITE DMA	110
ACCESSING A HARDWARE SPRITE	111
CHANGING THE APPEARANCE OF A SIMPLE SPRITE	112
MOVING A SIMPLE SPRITE	113
RELINQUISHING A SIMPLE SPRITE	120
Using VSprites	120
SPECIFYING THE SIZE OF A VSPRITE	121
SPECIFYING THE COLORS OF A VSPRITE	121
SPECIFYING THE SHAPE OF A VSPRITE	122
SPECIFYING VSPRITE POSITION	124
USING VSPRITE FLAGS	124
ADDING A VSPRITE	125

REMOVING A VSPRITE	126
GETTING THE VSPRITE LIST IN ORDER	126
DISPLAYING THE VSPRITES	127
VSPRITE OPERATIONS SUMMARY	129
VSPRITE ADVANCED TOPICS	131
Using Bobs	135
LINKING A BOB TO A VSPRITE STRUCTURE	135
SPECIFYING THE SIZE OF A BOB	136
SPECIFYING THE COLORS OF A BOB	136
SPECIFYING THE SHAPE OF A BOB	137
OTHER ITEMS INFLUENCING BOB COLORS	138
BOB PRIORITIES	141
SAVING THE PLAYFIELD DISPLAY	143
USING BOB FLAGS	144
ADDING A BOB	147
REMOVING A BOB	147
GETTING THE LIST OF BOBS IN ORDER	148
DISPLAYING BOBS	148
CHANGING BOBS	148
DOUBLE-BUFFERING	149
BOB OPERATIONS SUMMARY	151
BOB ADVANCED TOPICS	152
Topics Common to Both VSprites and Bobs	153
DETECTING GEL COLLISIONS	153
BOB/VSPRITE COLLISION BOUNDARIES WITHIN A RASTPORT	161
ADDING NEW FEATURES TO BOB/VSPRITE DATA STRUCTURES	161
Animation Structures and Controls	163
CHARACTERISTICS OF THE ANIMATION SYSTEM	163
KEEPING TRACK OF GRAPHIC OBJECTS	164
CLASSES OF ANIMATION OBJECTS	164
POSITIONS OF ANIMATION OBJECTS	165
ANIMATION TYPES	166
INITIALIZING THE ANIMATION SYSTEM	169
SPECIFYING THE ANIMATION OBJECTS	169
SPECIFYING ANIMATION COMPONENTS	170
DRAWING PRECEDENCE	172
ANIMATION SEQUENCING	173
SPECIFYING TIME FOR EACH IMAGE	174
YOUR OWN ANIMATION ROUTINE CALLS	176
MOVING THE OBJECTS	177
Complete Example Program	177

Chapter 4 Text	191
Introduction	191
Printing Text into a Drawing Area	192
CURSOR POSITION	192
BASELINE OF THE TEXT	193
SIZE OF THE FONT	194
PRINTING THE TEXT	194
SAMPLE PRINT ROUTINE	194
Selecting the Font	195
Selecting the Text Color	197
Selecting a Drawing Mode	197
Effects of Specifying Font Style	199
Adding a New Font to the System	200
Using a Disk Font	200
Finding Out Which Fonts Are Available	201
Contents of a Font Directory	201
The Disk Font	202
Defining a Font	203
THE TEXT NODE	203
FONT HEIGHT	203
FONT STYLE	203
FONT PREFERENCES	204
FONT WIDTH	204
FONT ACCESSORS	205
CHARACTERS REPRESENTED BY THIS FONT	205
THE CHARACTER DATA	206
A COMPLETE SAMPLE FONT	207
Sample Program	211

PART II

Chapter 5 Audio Device	221
Introduction	221
Definitions	223
Audio Functions and Commands	224
AUDIO AS A DEVICE	224
SCOPE OF COMMANDS	224
ALLOCATION AND ARBITRATION	225
PERFORMING AUDIO COMMANDS	226
COMMAND TYPES	226
SYSTEM FUNCTIONS	227
ALLOCATION/ARBITRATION COMMANDS	228
HARDWARE CONTROL COMMANDS	232
Example Programs	235
STEREO SOUND EXAMPLE	235
DOUBLE-BUFFERED SOUND SYNTHESIS EXAMPLE	240

Chapter 6 Timer Device	247
Introduction	247
Timer Device Units	248
Specifying the Time Request	248
Opening a Timer Device	249
Adding a Time Request	250
Closing a Timer	250
Additional Timer Functions and Commands	251
SYSTEM TIME	251
USING THE TIME ARITHMETIC ROUTINES	253
WHY USE TIME ARITHMETIC?	254
Sample Timer Program	254
Chapter 7 Trackdisk Device	261
Introduction	261
The Amiga Floppy Disk	262
Trackdisk Driver Commands	263
Creating an I/O Request	264
Opening a Trackdisk Device	266
Sending a Command to the Device	267
Terminating Access to the Device	267
Device-specific Commands	267
ETD_READ AND CMD_READ	267
ETD_WRITE AND CMD_WRITE	268
ETD_UPDATE AND CMD_UPDATE	268
ETD_CLEAR AND CMD_CLEAR	268
ETD_MOTOR AND TD_MOTOR	268
TD_FORMAT	269
TD_REMOVE	269
Status Commands	269
TD_CHANGENUM	269
TD_CHANGESTATE	270
TD_PROTSTATUS	270
Commands for Diagnostics and Repair	270
Trackdisk Driver Errors	270
Example Program	271
Chapter 8 Console Device	275
Introduction	275
System Functions	276
Console I/O	276
GENERAL CONSOLE SCREEN OUTPUT	276
CONSOLE KEYBOARD INPUT	277
Creating an I/O Request	277

Opening a Console Device	278
SENDING A CHARACTER STREAM TO THE CONSOLE DEVICE	279
Control Sequences for Screen Output	281
READING FROM THE CONSOLE	286
INFORMATION ABOUT THE READ-STREAM	287
CURSOR POSITION REPORT	288
WINDOW BOUNDS REPORT	289
SELECTING RAW INPUT EVENTS	289
Complex Input Event Reports	290
Keymapping	297
ABOUT QUALIFIERS	301
KEYTYPE TABLE ENTRIES	302
STRING-OUTPUT KEYS	303
CAPSABLE BIT TABLE	304
REPEATABLE BIT TABLE	305
DEFAULT LOW KEY MAP	305
DEFAULT HIGH KEY MAP	306
Closing a Console Device	307
Example Program	308
Chapter 9 Input Device	321
Introduction	322
Input Device Commands	322
IND_ADDHANDLER COMMAND	324
IND_REMHANDLER COMMAND	326
IND_WRITEEVENT COMMAND	326
IND_SETTHRESH COMMAND	327
IND_SETPERIOD COMMAND	327
Input Device and Intuition	327
Sample Program	328
Chapter 10 Keyboard Device	337
Introduction	337
Keyboard Device Commands	338
KBD_ADDRESETHANDLER	339
KBD_REMRESETHANDLER	340
KBD_RESETHANDLERDONE	340
KBD_READMATRIX	340
KBD_READEVENT	341
Example Keyboard Read-event Program	342

Chapter 11 Gameport Device	345
Introduction	345
Gameport Device Commands	346
GPD_SETCTYPE	346
GPD_GETCTYPE	347
GPD_SETTRIGGER	348
Example Programs	349
MOUSE PROGRAM	349
JOYSTICK PROGRAM	355
Chapter 12 Narrator Device	361
Introduction	362
The Translator Library	362
USING THE TRANSLATE FUNCTION	362
ADDITIONAL NOTES ABOUT TRANSLATE	363
The Narrator Device	363
OPENING THE NARRATOR DEVICE	363
CONTENTS OF THE WRITE REQUEST BLOCK	364
CONTENTS OF THE READ REQUEST	365
SYNCHRONIZING NARRATOR READS	366
PERFORMING A WRITE AND A READ	366
Sample Program	367
How to Write Phonetically for Narrator	372
PHONETIC SPELLING	373
CHOOSING THE RIGHT VOWEL	373
CHOOSING THE RIGHT CONSONANT	374
CONTRACTIONS AND SPECIAL SYMBOLS	374
STRESS AND INTONATION	375
HOW AND WHERE TO PUT THE STRESS MARKS	375
WHAT STRESS VALUE DO I USE?	376
PUNCTUATION	377
HINTS FOR INTELLIGIBILITY	378
EXAMPLE OF ENGLISH AND PHONETIC TEXTS	378
CONCLUDING REMARKS	379
The More Technical Explanation	379
Table of Phonemes	380
Chapter 13 Serial Device	383
Introduction	383
Opening the Serial Device	384
Reading from the Serial Device	385
FIRST ALTERNATIVE MODE FOR READING	386
SECOND ALTERNATIVE MODE FOR READING	387
TERMINATION OF THE READ	388
Writing to the Serial Device	388

Setting Serial Parameters	390
SERIAL FLAGS	391
SETTING THE PARAMETERS	393
Errors from the Serial Device	393
Closing the Serial Device	394
Example Program	395
Chapter 14 Parallel Device	401
Introduction	401
Opening the Parallel Device	402
Reading from the Parallel Device	403
ALTERNATIVE MODE FOR READING	404
TERMINATION OF THE READ	405
Writing to the Parallel Device	405
Setting Parallel Parameters	406
PARALLEL FLAGS	406
SETTING THE PARAMETERS	407
Errors from the Parallel Device	407
Closing the Parallel Device	408
Example Program	409
Chapter 15 Printer Device	413
Introduction	413
PRT:—THE AMIGADOS PRINTER DEVICE	414
SER:—THE AMIGADOS SERIAL DEVICE	414
PAR:—THE AMIGADOS PARALLEL DEVICE	414
THE PRINTER DEVICE	415
Printer Device Output	415
Opening the AmigaDOS Printer Device	415
Data Structures Used During Printer I/O	416
Creating an I/O Request	417
Opening a Printer Device	418
Writing to the Printer	418
PRINTER COMMAND DEFINITIONS	419
Transmitting a Command to the Printer Device	422
Dumping a RastPort to the Printer	423
ADDITIONAL NOTES ABOUT GRAPHICS DUMPS	427
Creating a Printer Driver	428
SAMPLE CODE	431
WRITING A GRAPHICS PRINTER DRIVER	431
WRITING AN ALPHANUMERIC PRINTER DRIVER	435
Chapter 16 Clipboard Device	439
Introduction	439
Clipboard Commands	440

Clipboard Data	441
Clipboard Messages	442
Multiple Clips	442
Example Program	443
Support Functions Called from Example Program	447
 PART III	
Chapter 17 Math Functions	453
Introduction	453
FFP Floating Point Data Format	454
FFP Basic Mathematics Library	455
FFP Transcendental Mathematics Library	461
FFP Mathematics Conversion Library	469
IEEE Double-precision Basic Math Library	472
 Chapter 18 Workbench	 479
Introduction	479
The Icon Library	480
The Info File	481
THE DISKOBJECT STRUCTURE	481
THE GADGET STRUCTURE	483
ICONS WITH NO POSITION	484
Workbench Environment	484
START-UP MESSAGE	485
THE STANDARD START-UP CODE	486
The ToolTypes Array	487
Example Programs	488
FRIENDLYTOOL	488
START-UP PROGRAM	489
ECHO.C	497
 Appendix A Library Summaries	 A-1
 Appendix B Device Summaries	 B-1
 Appendix C Resource Summaries	 C-1
 Appendix D Include Files	 D-1
C Include Files—“.h” Files	
Assembly-language Include Files—“.i” Files	
 Appendix E Printer Device Source Code	 E-1
 Appendix F Skeleton Device/Library Code	 F-1
 Index	 Index-1

PART I

Chapter 1

Graphics Primitives

This chapter describes the basic graphics tools. It covers the graphics support structures, display routines, and drawing routines. Many of the operations described in this section are also performed by the Intuition software. See the book called *Intuition: The Amiga User Interface* for more information.

Introduction

The Amiga has two basic types of graphics support routines: display routines and drawing routines. These routines are very versatile and allow you to define any combination of drawing and display area you may wish to use.

The first section of this chapter defines the display routines. These routines show you how to form and manipulate a display, including the following aspects of display use:

- o How to identify the memory area that you wish to have displayed
- o How to position the display area window to show only a certain portion of a larger drawing area
- o How to split the screen into as many vertically stacked slices as you wish
- o Whether to use high-resolution (640 pixels across) or low-resolution (320 pixels across) display mode for a particular screen segment, and whether to use interlaced (400 lines top to bottom) or non-interlaced (200 lines) mode
- o How to specify how many color choices per pixel are to be available in a specific section of the display

The next section of the chapter explains all of the available modes of drawing supported by the system software, including how to do the following:

- o Reserve memory space for use by the drawing routines
- o Define the colors that can be drawn into a drawing area
- o Define the colors of the drawing pens (foreground pen, background pen for patterns, and outline pen for area-fill outlines)
- o Define the pen position in the drawing area
- o Draw lines, define vertex points for area-filling, and specify the area-fill color and pattern
- o Define a pattern for patterned line drawing
- o Change drawing modes

- o Read or write individual pixels in a drawing area
- o Copy rectangular blocks of drawing area data from one drawing area to another
- o Use a template (predefined shape) to draw an object into a drawing area

COMPONENTS OF A DISPLAY

In producing a display, you are concerned with two primary components: sprites and the playfield. Sprites are the easily movable parts of the display. The playfield is the static part of the display and forms a backdrop against which the sprites can move and with which the sprites can interact.

This chapter covers the creation of the background. Sprites are described in chapter 3, "Animation."

INTRODUCTION TO RASTER DISPLAYS

The Amiga produces its video displays on standard television or video monitors by using raster display techniques. The picture you see on the video display screen is made up of a series of horizontal video lines stacked one on top of another, as illustrated in figure 1-1. Each line represents one sweep of an electronic video beam, which "paints" the picture as it moves along. The beam sweeps from left to right, producing the full screen one line at a time. After producing the full screen, the beam returns to the top of the display screen.

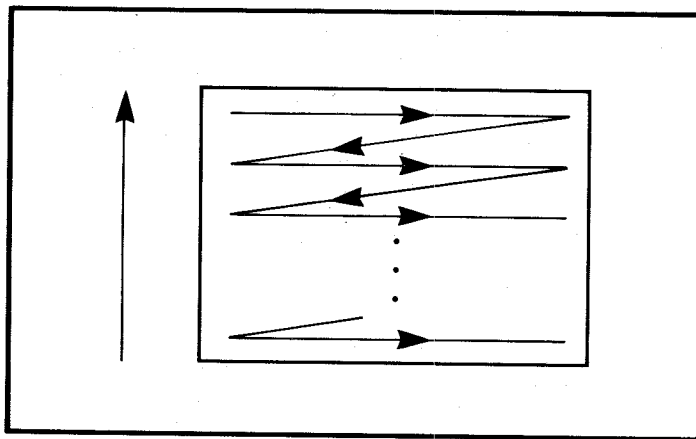


Figure 1-1: How the Video Display Picture Is Produced

The diagonal lines in the figure show how the video beam returns to the start of each horizontal line.

Effect of Display Overscan on the Viewing Area

To assure that the picture entirely fills the viewable region of the screen, the manufacturer of the video display usually creates a deliberate *overscan*. That is, the video beam is swept across a larger section than the front face of the screen can actually display. The video beam actually covers 262 vertical lines. The user, however, sees only the portion of the picture that is within the center region of the display, which is about 200 rows, as illustrated in figure 1-2 below. The graphics system software lets you specify more than 200 rows.

Overscan also restricts the amount of video data that can appear on each display line. The system software allows you to specify a display width of up to 352 pixels (or 704 in high-resolution mode) per horizontal line. Generally, however, you should use the standard values of 320 (or 640 in high-resolution mode) for most applications.

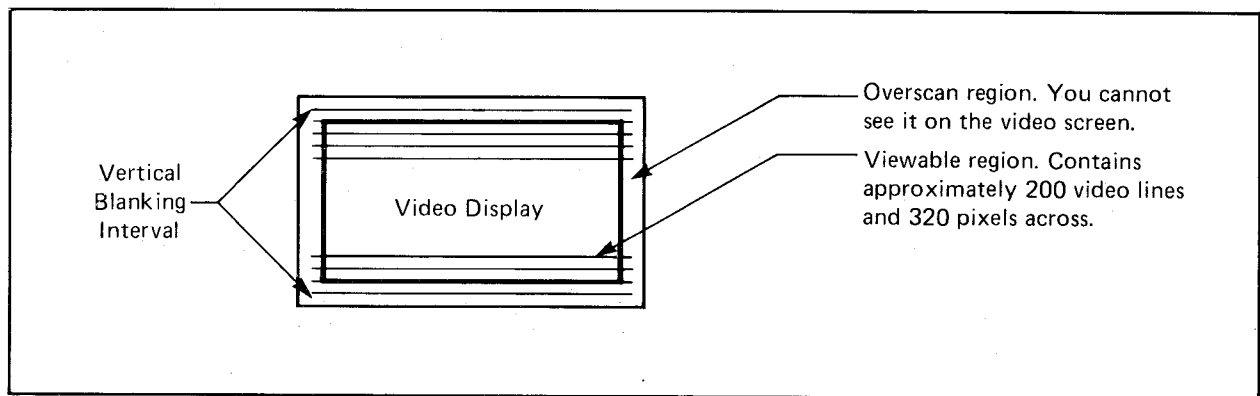


Figure 1-2: Display Overscan Restricts Usable Picture Area

The time during which the video beam is in the region below the bottom line of the viewable area and above the top line of the next display field is called the *vertical blanking interval*.

Color Information for the Video Lines

The hardware reads the system display memory to obtain the color information for each line. As the video display beam sweeps across the screen producing the display line, it changes color, producing the images you have defined.

INTERLACED AND NON-INTERLACED MODES

In producing the complete display (262 video lines), the video display device produces the top line, then the next lower line, then the next, until it reaches the bottom of the screen. When it reaches the bottom, it returns to the top to start a new scan of the screen. Each complete set of 262 lines is called a *display field*. It takes about 1/60th of a second to produce a complete display field.

The Amiga has two vertical display modes: *interlaced* and *non-interlaced*. In non-interlaced mode, the video display produces the same picture for each successive display field. A non-interlaced display normally has about 200 lines in the viewable area (for a full-screen size display).

To make the display more precise in the vertical direction, you use interlaced mode, which displays twice as much data in the same vertical area as non-interlaced mode. Within the same amount of viewable area, you can display 400 video lines instead of 200.

For interlaced mode, the video beam scans the screen at the same rate (1/60th of a second per complete video display field); however, it takes two display fields to form a complete video display picture. During the first of each pair of display fields, the system hardware shows the odd-numbered lines of an interlaced display (1, 3, 5, and so on). During the second display field, it shows the even-numbered lines (2, 4, 6 and so on). These sets of lines are taken from data defining 400 lines. During the display, the hardware moves the second display field's lines downward slightly from the position of the first, so that the lines in the second field are "interlaced" with those of the first field, giving the higher vertical resolution of this mode. For an interlaced display, the data in memory defines twice as many lines as for a non-interlaced display, as shown in figure 1-3.

	DATA AS DISPLAYED	DATA IN MEMORY
Odd field	— Line 1	Line 1
Even field	— Line 1	Line 2
Odd field	— Line 2	Line 3
Even field	— Line 2	Line 4
	⋮	⋮
	⋮	⋮
Odd field	— Last line	Line 399
Even field	— Last line	Line 400

Figure 1-3: Interlaced Mode — Display Fields and Data in Memory

Figure 1-4 shows a display formed as display lines 1, 2, 3, 4, ... 400. The 400-line interlaced display uses the same physical display area as a 200-line non-interlaced display.

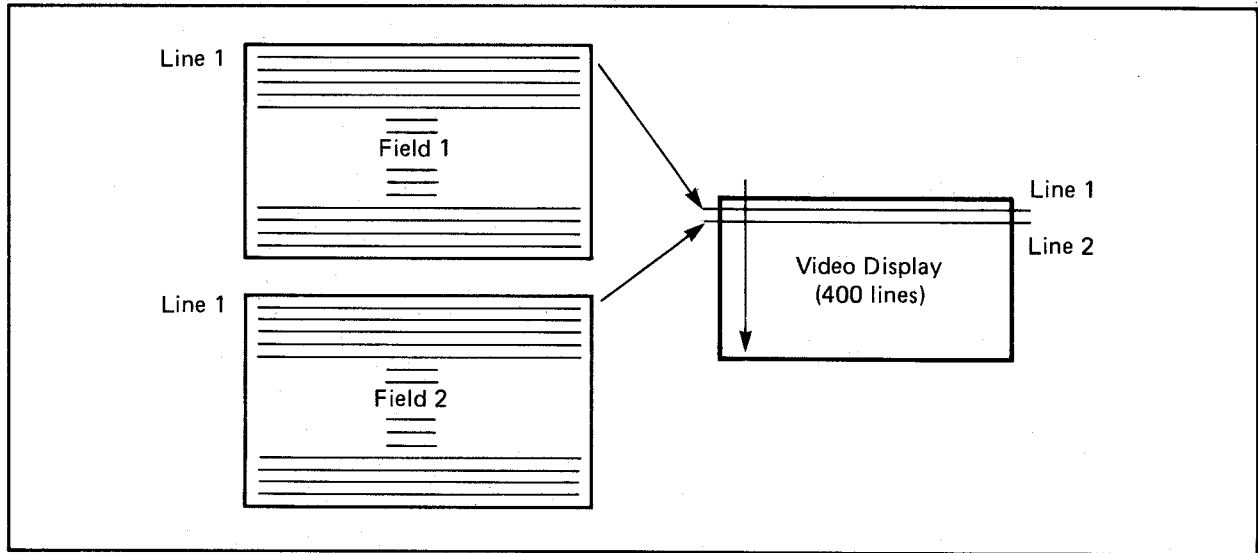


Figure 1-4: Interlaced Mode Doubles Vertical Resolution

During an interlaced display, it appears that both display fields are present on the screen at the same time and form one complete picture. This phenomenon is called *video persistence*.

HIGH- AND LOW-RESOLUTION MODES

The Amiga also has two horizontal display modes: *high-resolution* and *low-resolution*. High-resolution mode provides (nominally) 640 distinct pixels (picture elements) across a horizontal line. Low-resolution provides (nominally) 320 pixels across each line. Low-resolution mode allows up to 32 colors at one time, and high-resolution mode allows 16 colors (out of 4,096 choices) at one time.

One other display mode affects the number of colors you can display at one time: *hold-and-modify*. Hold-and-modify mode allows you to display all 4,096 colors on the screen at once.

FORMING AN IMAGE

To create an image, you write data (that is, you “draw”) into a memory area in the computer. From this memory area, the system can retrieve the image for display. You tell the system exactly how the memory area is organized, so that the display is correctly produced. You use a block of memory words at sequentially increasing addresses to represent a rectangular region of

data bits. Figure 1-5 shows the contents of three example memory words: 0 bits are shown as blank rectangles, and 1 bits as filled-in rectangles.

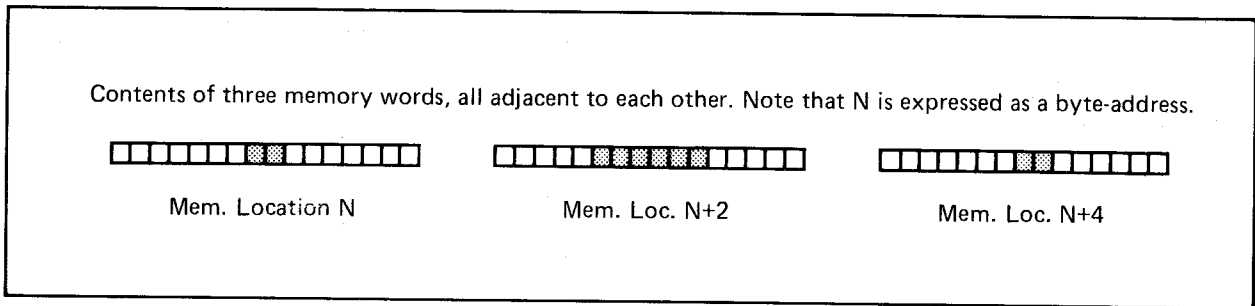


Figure 1-5: Sample Memory Words

The system software lets you define linear memory as rectangular regions, called *bit-planes*. Figure 1-6 shows how the system views the same three words as a bit-plane, wherein the data bits form an x-y plane.

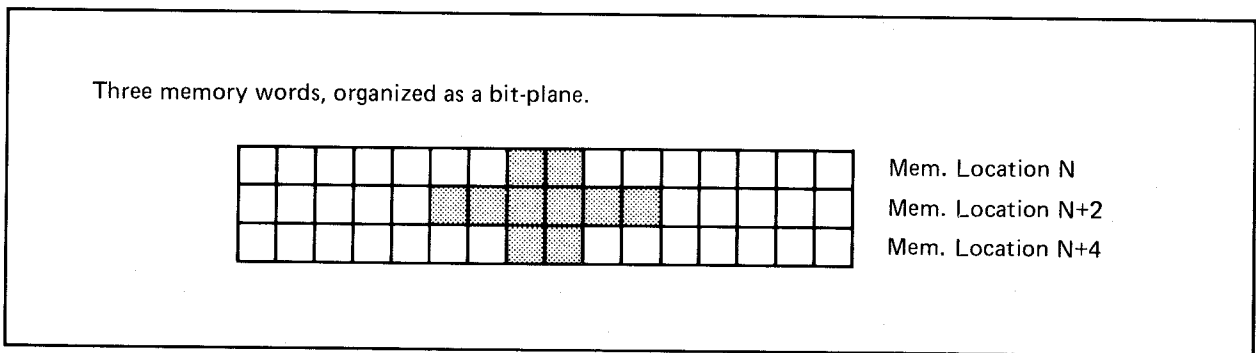


Figure 1-6: A Rectangular “Look” at the Sample Memory Words

Figure 1-7 shows how 4,000 words (8,000 bytes) of memory can be organized to provide enough bits to define a single bit-plane of a full-screen, low-resolution video display (320 x 200).

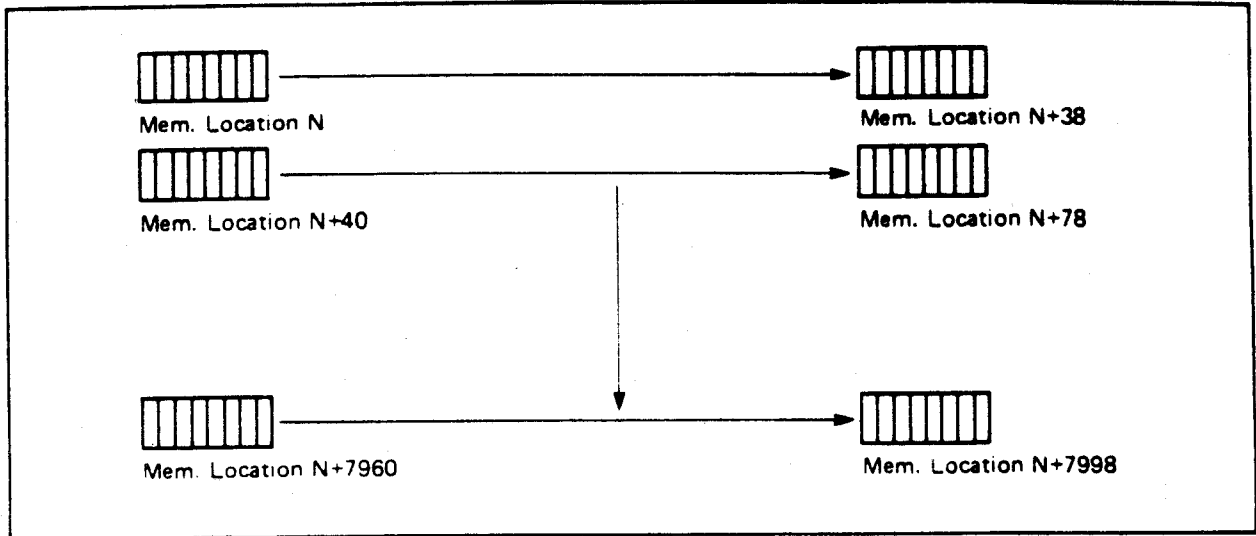


Figure 1-7: Bit-Plane for a Full-screen, Low-resolution Display

Each memory data word contains 16 data bits. The color of each pixel on a video display line is directly related to the value of one or more data bits in memory, as follows:

- o If you create a display in which each pixel is related to only one data bit, you can only select from only two possible colors, because each bit can have a value of only 0 or 1.
- o If you use two bits per pixel, there is a choice of four different colors because there are four possible combinations of the values of 0 and 1 from each of the two bits.
- o If you specify three, four, or five bits per pixel, you will have eight, sixteen, or thirty-two possible choices of a color for each pixel.

To create multicolored images, you must tell the system how many bits are to be used per pixel. The number of bits per pixel is the same as the number of bit-planes used to define the image.

As the video beam sweeps across the screen, the system retrieves one data bit from each bit-plane. Each of the data bits is taken from a different bit-plane, and one or more bit-planes are used to fully define the video display screen. For each pixel, data-bits in the same x,y position in each bit-plane are combined by the system hardware to create a binary value. This value determines the color that appears on the video display for that pixel. (See figure 1-8.)

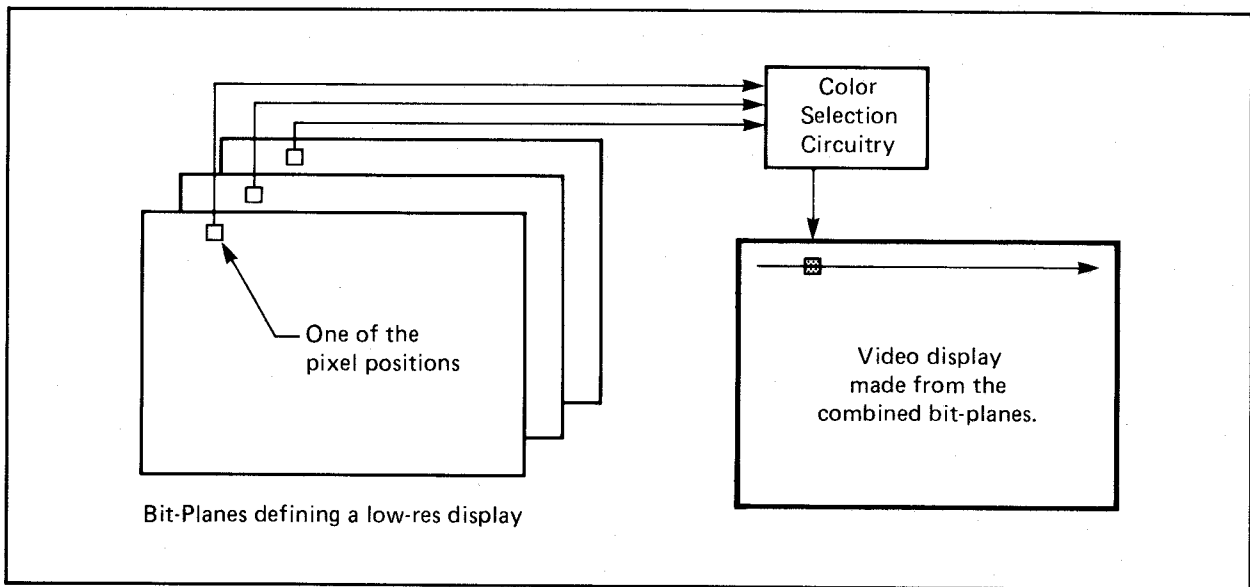


Figure 1-8: Bits from Each Bit-Plane Select Pixel Color

You will find more information showing how the data bits actually select the color of the displayed pixel in the section called “ViewPort Color Selection.”

ROLE OF THE COPPER (COPROCESSOR)

The Amiga has a special-purpose coprocessor, called the *Copper*, that can control nearly the entire graphics system. The Copper can control register updates, reposition sprites, change the color palette, and update the blitter. The graphics and animation routines use the Copper to set up lists of instructions for handling displays, and advanced users can write their own “user Copper lists.”

Display Routines and Structures

Caution: This section describes the lowest-level graphics interface to the system hardware. If you use any of the routines and the data structures described in these sections, your program will essentially take over the entire display. It will not, therefore, be compatible with the multiwindow operating environment, known as Intuition, which is used by AmigaDOS.

The descriptions of the display routines, as well as those of the drawing routines, occasionally use the same terminology as that in *Intuition: The Amiga User Interface*. These routines and data structures are the same ones that Intuition software uses to produce its displays.

The computer produces a display from a set of instructions you define. You organize the instructions as a set of parameters known as the **View** structure. Figure 1-9 shows how the system interprets the contents of a **View** structure. This drawing shows a complete display composed of two different component parts, which could, for example, be a low-resolution, multicolored part and a high-resolution, two-colored part.

A complete display consists of one or more **ViewPorts**, whose display sections are separated from each other by at least one blank line. The viewable area defined by each **ViewPort** is a rectangular cut from the same size (or larger) raster. You are essentially defining a display consisting of a number of vertically stacked display areas in which separate sections of graphics rasters can be shown.

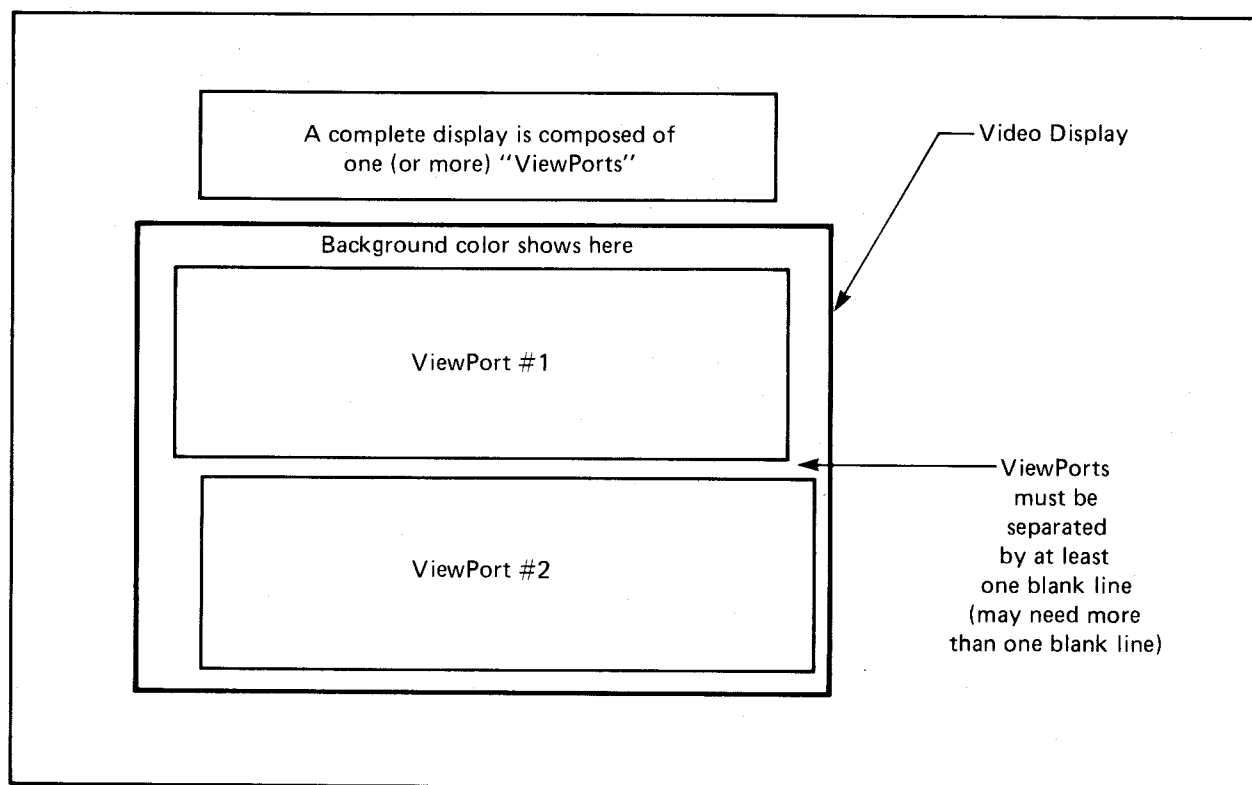


Figure 1-9: The Display Is Composed of ViewPorts

LIMITATIONS ON THE USE OF VIEWPORTS

The system software for defining **ViewPorts** allows only vertically stacked fields to be defined. Figure 1-10 shows acceptable and unacceptable display configurations. If you want to create overlapping windows, define a single **ViewPort** and manage the windows yourself within that **ViewPort**.

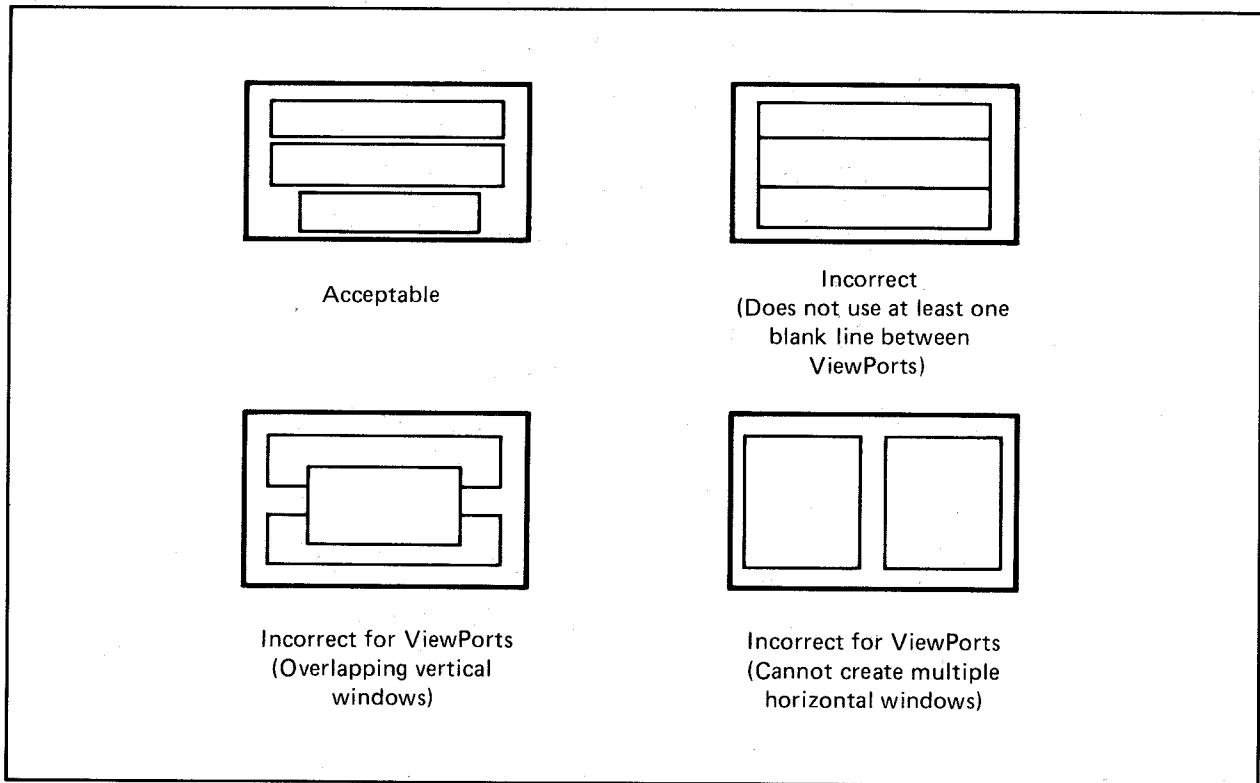


Figure 1-10: Correct and Incorrect Uses of ViewPorts

A **ViewPort** is related to the custom screen option of Intuition. In a custom screen, you can split the screen into slices as shown in the “correct” illustration of figure 1-10. Each custom screen can have its own set of colors, use its own resolution, and show its own display area. Within a **ViewPort**—actually within its associated **RastPort** (drawing area definition)—it is possible to split the display into separate drawing areas called *windows*. The **ViewPort** is simply an indivisible window into a possibly larger complex drawing area.

CHARACTERISTICS OF A VIEWPORT

To describe a **ViewPort** fully, you need to set the following parameters: height, width, and display mode.

In addition to these parameters, you must also tell the system the location in memory from which the data for the **ViewPort** display should be retrieved, and how to position the final **ViewPort** display on the screen.

VIEWPORT SIZE SPECIFICATIONS

Figure 1-11 illustrates that the variables **DHeight**, and **DWidth** specify the size of a **ViewPort**.

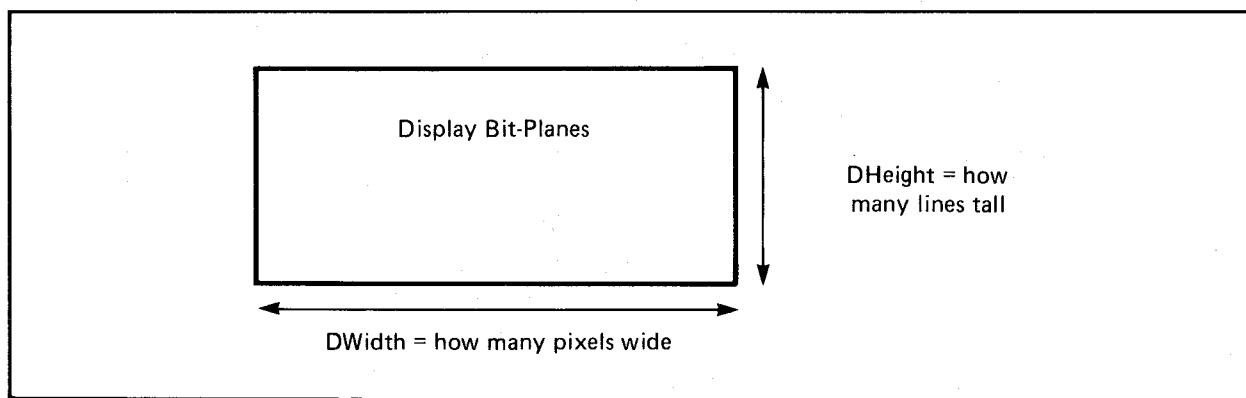


Figure 1-11: Size Definition for a ViewPort

ViewPort Height

The variable **DHeight** determines how many video lines will be reserved to show the height of this display segment. The size of the actual segment depends on whether you define a non-interlaced or an interlaced display. An interlaced display is half as tall as a non-interlaced display of the same number of lines.

For example, a **View** consisting of two **ViewPorts** might be defined as follows:

- o **ViewPort #1** is 150 lines, high-resolution mode (uses the top three-quarters of the display).

- o **ViewPort #2** is 49 lines of low-resolution mode (uses the bottom quarter of the display and allows the space for the required blank line between **ViewPorts**).

The user interface software (Intuition) assumes a standard configuration of 200 rows (400 in interlaced mode).

ViewPort Width

The **DWidth** variable determines how wide, in current pixels, the display segment will be. If you are using low-resolution mode, you should specify a width of 320 pixels per horizontal line. If you are using high-resolution mode, you should specify a width of 640 pixels. You may specify a smaller value of pixels per line to produce a narrower display segment.

Although the system software allows you define low-resolution displays as wide as 352 pixels and high-resolution displays as wide as 704 pixels, you should not exceed the normal values of 320 or 640, respectively. Because of display overscan, many video displays will not be able to show all of a wider display, and sprite display may be affected. If you are using hardware sprites or VSprites with your display, and you specify **ViewPort** widths exceeding 320 or 640 pixels (for low- or high-resolution, respectively), it is likely that hardware sprites 5, 6, and 7 will not be rendered on the screen. These sprites may not be rendered because playfield DMA (direct memory access) takes precedence over sprite DMA when an extra-wide display is produced.

VIEWPORT COLOR SELECTION

The maximum number of colors that a **ViewPort** can display is determined by the depth of the **BitMap** that the **ViewPort** displays. The depth is specified when the **BitMap** is initialized. See the section below called "Preparing the BitMap Structure."

Depth determines the number of bit-planes used to define the colors of the rectangular image you are trying to build (the raster image) and the number of different colors that can be displayed at the same time within a **ViewPort**. For any single pixel, the system can display any one of 4,096 possible colors.

Table 1-1 shows depth values and the corresponding number of possible colors for each value.

Table 1-1: Depth Values and Number of Colors in the ViewPort

Colors	Depth Value	
2	1	
4	2	
8	3	
16	4	(Note 1)
32	5	(Notes 1,2)
4,096	6	(Notes 1,2,3)
32	6	(Notes 1,2)

Notes:

1. Single-playfield mode only — **ViewPort** mode not DUALPF
2. Low-resolution mode only — **ViewPort** mode not HIRES
3. Hold-and-modify mode only — **ViewPort** mode = HAM

The color palette used by a **ViewPort** is specified in a **ColorMap**. See the section called “Preparing the ColorMap” for more information.

Depending on whether single- or dual-playfield mode is used, the system will use different color register groupings for interpreting the on-screen colors. Table 1-2 below details how the depth and the **Modes** variable in the **ViewPort** structure affect the registers the system uses.

Table 1-2: Single-playfield Mode (**Modes** variable not equal to DUALPF)

Depth	Color Registers Used	
1	0,1	
2	0-3	
3	0-7	
4	0-15	
5	0-31	
6	0-16	(if modes = HAM)

Table 1-3 shows the five possible combinations when the **Modes** variable is set to DUALPF.

Table 1-3: Dual-playfield Mode (**Modes** variable = DUALPF)

Depth (PF-1)	Color Registers	Depth (PF-2)	Color Registers
1	0,1	1	8,9
2	0-3	1	8,9
2	0-3	2	8-11
3	0-7	2	8-11
3	0-7	3	8-15

The system has seven different display modes that you can specify for each **ViewPort**. The seven bits that control the modes are DUALPF, PFBA, HIRES, LACE, HAM, SPRITES, and VP_HIDE. A mode becomes active if you set the corresponding bit to 1 in the **Modes** variable of the **ViewPort** structure. After you initialize the **ViewPort**, you can set the bit(s) for the modes you want. (See the section called “Preparing the ViewPort Structure” for more information about initializing a **ViewPort**.)

Modes DUALPF and PFBA are related. DUALPF tells the system to treat the raster specified by this **ViewPort** as the first of two independent and separately controllable playfields. It also modifies the manner in which the pixel colors are selected for this raster.

When PFBA is a 1, it specifies that a second playfield has video priority over the first one. Playfield relative priorities can be controlled when the playfield is split into two overlapping regions. Single-playfield and dual-playfield modes are discussed in “Advanced Topics” below.

HIRES tells the system that the raster specified by this **ViewPort** is to be displayed with 640 horizontal pixels rather than 320 horizontal pixels.

LACE tells the system that the raster specified by this **ViewPort** is to be displayed in interlaced mode. If the **ViewPort** is non-interlaced and the **View** is interlaced, the **ViewPort** will be displayed at its specified height and will look only slightly different than it would look when displayed in a non-interlaced **View**. See “Interlaced Mode versus Non-interlaced Mode” below for more information.

HAM tells the system to use “hold-and-modify” mode, a special mode that lets you display up to 4,096 colors on screen at the same time. It is described in the “Advanced Topics” section.

SPRITES tells the system that you are using sprites in this display (either VSprites or Simple Sprites). This bit, when a 1, tells the software to load color registers for sprites. See chapter 3, "Animation," for more information about sprites.

VP_HIDE tells the system that this **ViewPort** is obscured by other **ViewPorts**. When a **View** is constructed, no display instructions are generated for this **ViewPort**.

EXTRA_HALFBRITE is reserved for future use.

Single-playfield Mode versus Dual-playfield Mode

When you specify single-playfield mode (see figure 1-12), you are asking that the system treat all bit-planes as part of the definition of a single playfield image. Each of the bit-planes defined as part of this **ViewPort** contributes data bits that determine the color of the pixels in a single playfield.

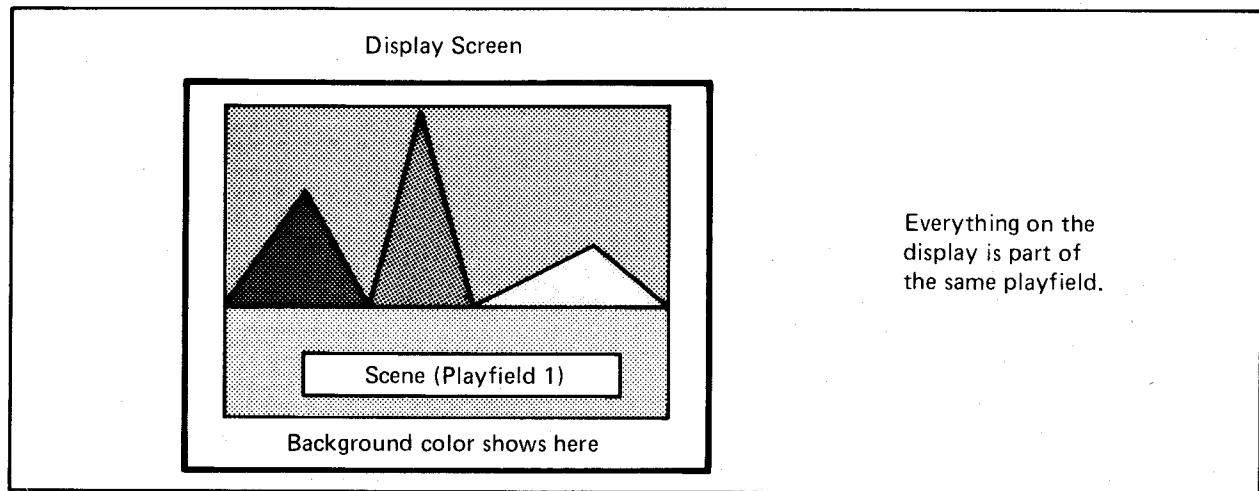


Figure 1-12: A Single-playfield Display

If you use dual-playfield mode (**ViewPort.Modes = DUALPF**), you can define two independent, separately controllable playfield areas (see figure 1-13).

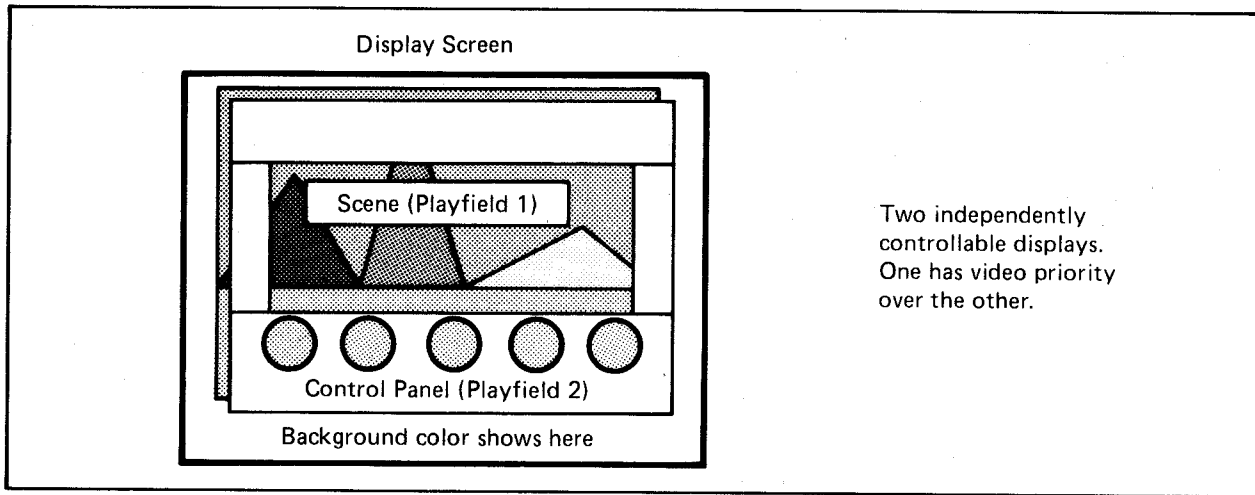


Figure 1-13: A Dual-playfield Display

In figure 1-13, the display mode bit PFBA is set to 1. If $PFBA = 0$, the relative priorities will be reversed; playfield 2 will appear to be behind playfield 1.

Low-resolution Mode versus High-resolution Mode

In low-resolution mode, horizontal lines of 320 pixels fill most of the ordinary viewing area. The system software lets you define a screen segment width up to 352 pixels in this mode, or you can define a screen segment as narrow as you desire. In high-resolution mode (also called "normal" resolution), 640 pixels fill a horizontal line. In this mode you can specify any width from 0 to 704 pixels. Overscan normally limits you to showing only 0 to 320 pixels per line in low-resolution mode or 0 to 640 pixels per line in high-resolution mode. Intuition assumes the nominal 320-pixel or 640-pixel width (see figure 1-14).

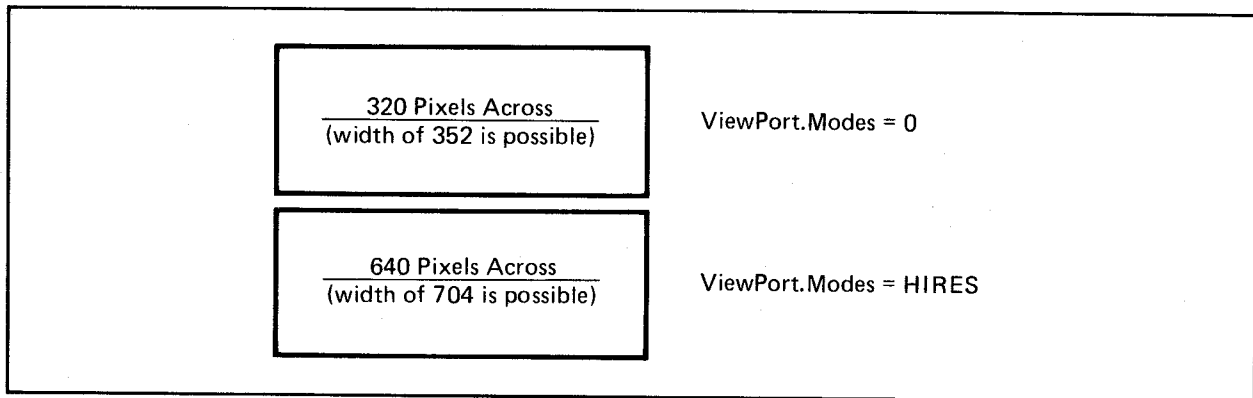


Figure 1-14: How HIRES Affects Width of Pixels

Interlaced Mode versus Non-interlaced Mode

In interlaced mode, there are twice as many lines available as in non-interlaced mode, providing better vertical resolution in the same display area (see figure 1-15).

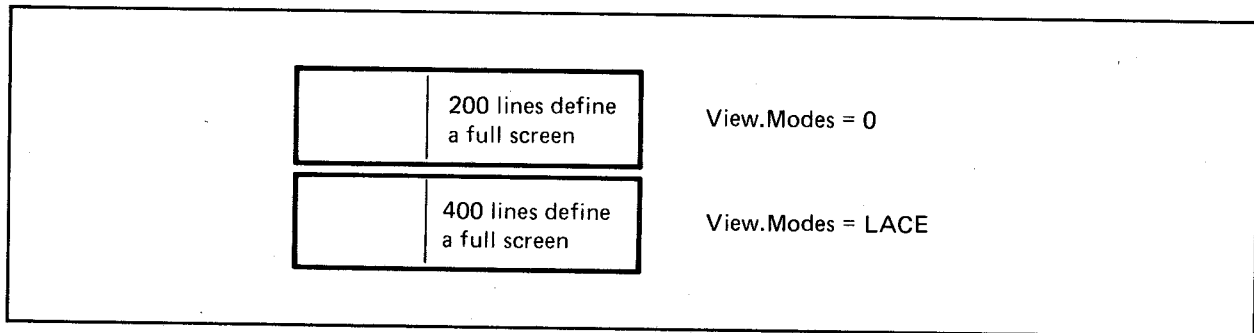


Figure 1-15: How LACE Affects Vertical Resolution

If the **View** structure does not specify LACE, and the **ViewPort** specifies LACE, you may see only every other line of the **ViewPort** data. If the **View** structure specifies LACE and the **ViewPort** is non-interlaced, the same **ViewPort** data will be repeated in both fields. The height of the **ViewPort** display is the height specified in the **ViewPort** structure. If both the **View** and the **ViewPort** are interlaced, the **ViewPort** will be built with double the normal vertical resolution. That means it will need twice as much data space in memory as a non-interlaced picture for this display.

VIEWPORT DISPLAY MEMORY

The picture you create in memory can be larger than the screen image that can be displayed within your **ViewPort**. This big picture (called a raster and represented by the **BitMap** structure) can have a maximum size of 1,024 by 1,024 pixels. Because a picture this large cannot fit fully on the display, you specify which piece of it to display. Once you have selected the piece to be shown, you can specify where it is to appear on the screen.

The example in figure 1-16 introduces terms that tell the system how to find the display data and how to display it in the **ViewPort**. These terms are **RHeight**, **RWidth**, **RyOffset**, **RxOffset**, **DHeight**, **DWidth**, **DyOffset** and **DxOffset**.

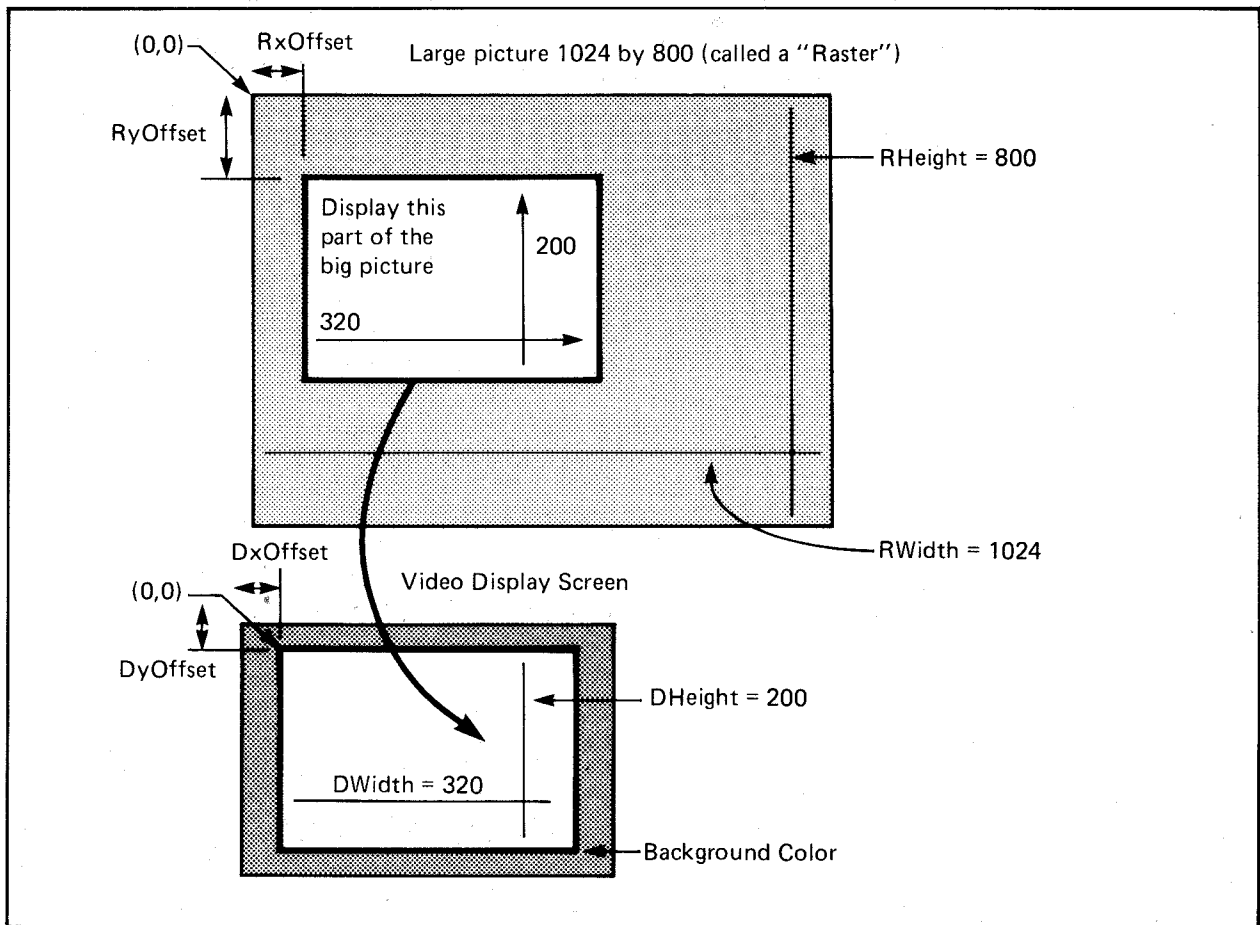


Figure 1-16: ViewPort Data Area Parameters

The terms **RHeight** and **RWidth** do not appear in actual system data structures. They refer to the dimensions of the raster and are used here to relate the size of the raster to the size of the display area. **RHeight** is the number of rows in the raster, and **RWidth** is the number of

bytes per row times 8. The raster shown in the figure is too big to fit entirely in the display area, so you tell the system which pixel of the raster should appear in the upper left corner of the display segment specified by your **ViewPort**. The variables that control that placement are **RyOffset** and **RxOffset**.

To compute **RyOffset** and **RxOffset**, you need **RHeight**, **RWidth**, **DHeight**, and **DWidth**. The **DHeight** and **DWidth** variables define the height and width in pixels of the portion of the display that you want to appear in the **ViewPort**. The example shows a full-screen, low-resolution mode (320-pixel), non-interlaced (200-line) display formed from the larger overall picture.

Normal values for **RyOffset** and **RxOffset** are defined by the formulas:

$$0 \leq \text{RyOffset} \leq (\text{RHeight} - \text{DHeight})$$
$$0 \leq \text{RxOffset} \leq (\text{RWidth} - \text{DWidth})$$

Once you have defined the size of the raster and the section of that raster that you wish to display, you need only specify where to put this **ViewPort** on the screen. This is controlled by the variables **DyOffset** and **DxOffset**. A value of 0 for each of these offsets places a normal-sized picture in a centered position at the top, bottom, left and right on the display screen. Possible values for **DyOffset** range from -16 to +200 (-32 to +400 if **View.Modes** includes LACE). Possible values for **DxOffset** range from -16 to +352 (-32 to +704 if **ViewPort.Modes** includes HIRES).

The parameters shown in the figure above are distributed in the following data structures:

- o **RasInfo** (information about the raster) contains the variables **RxOffset** and **RyOffset**. It also contains a pointer to the **BitMap** structure.
- o **View** (information about the whole display) includes the variables that you use to position the whole display on the screen. The **View** structure contains a **Modes** variable used to determine if the whole display is to be interlaced or non-interlaced. It also contains pointers to its list of **ViewPorts** and pointers to the Copper instructions produced by the system to create the display you have defined.
- o **ViewPort** (information about this piece of the display) includes the values **DxOffset** and **DyOffset** that are used to position this slice relative to the overall **View**. The **ViewPort** also contains the variables **DHeight** and **DWidth**, which define the size of this slice; a **Modes** variable; and a pointer to the local **ColorMap**. Each **ViewPort** also contains a pointer to the next **ViewPort**. You create a linked list of **ViewPorts** to define the complete display.
- o **BitMap** (information about memory usage) tells the system where to find the display and drawing area memory and shows how this memory space is organized.

You must allocate enough memory for the display you define. The memory you use for the display may be shared with the area control structures used for drawing. This allows you to draw into the same areas that you are currently displaying on the screen.

As an alternative, you can define two **BitMaps**. One of them can be the active structure (that being displayed) and the other can be the inactive structure. If you draw into one **BitMap** while displaying another, the user cannot see the drawing taking place. This is called *double-buffering* of the display. See “Advanced Topics” below for an explanation of the steps required for double-buffering. Double-buffering takes twice as much memory as single-buffering because two full displays are produced.

To determine the amount of required memory for each **ViewPort** for single-buffering, you can use the following formula.

$$\text{bytes_per_ViewPort} = \text{Depth} * \text{RASSIZE}(\text{Width}, \text{Height});$$

RASSIZE is a system macro attuned to the current design of the system memory allocation for display rasters. See *graphics/gfxmacros.h* for the formula with which **RASSIZE** is calculated.

For example, a 32-color **ViewPort** (depth = 5), 320 pixels wide by 200 lines high uses 40,000 bytes (as of this writing). A 16-color **ViewPort** (depth = 4), 640 pixels wide by 400 lines high uses 128,000 bytes (as of this writing).

FORMING A BASIC DISPLAY

This section offers an example that shows how to create a single **ViewPort** with a size of 200 lines, in which the area displayed is the same size as the big picture (raster) stored in memory. The example also shows how this **ViewPort** becomes the single display segment of a **View** structure. Following the description of the individual operations, the “Graphics Example Program” section pulls all of the pieces into a complete executable program. Instead of linking these routines to drawing routines, the example allocates memory specifically and only for the display (instead of sharing the memory with the drawing routines) and writes data directly to this memory. This keeps the display and the drawing routines separate for purposes of discussion.

Here are the data structures that you need to define to create a basic display:

```
struct View v;           /* The name used here for a View is v,
struct ViewPort vp;     * for a ViewPort is vp,
struct BitMap b;       * for a BitMap is b,
struct RasInfo ri;     * and for a RasInfo is ri. */
```


Opening the Graphics Library

Most of the system routines used here are located in the graphics library. When you compile your program, you must provide a way to tell the compiler to link your calling sequences into the routine library in which they are located. You accomplish this by declaring the variable called **GfxBase**. Then, by opening the graphics library, you provide the value (address of the library) that the system needs for linking with your program. See the “Libraries” chapter in the *Amiga ROM Kernel Reference Manual: Exec* for more information.

Here is a typical sequence:

```
struct GfxBase *GfxBase;    /* declare the name *GfxBase as a
                           * pointer to the corresponding library */
```

Preparing the View Structure

The following code section prepares the **View** structure for further use:

```
InitView( &v );           /* initialize the View structure */
v.ViewPort = &vp; /* tell the View structure where to find the
                   * first ViewPort in a possible list of Viewports */
```

Preparing the ViewPort Structure

The following code section prepares the **ViewPort** structure for further use:

```
InitVPort( &vp );        /* initialize the structure (set up default values) */

vp.DWidth = WIDTH; /* how wide is the display */
vp.DHeight = HEIGHT; /* how tall is the display for this ViewPort */
vp.RasInfo = &ri;     /* pointer to a RasInfo structure */
vp.ColorMap = GetColorMap(32); /* using a 32-color map */
```

The **InitVPort()** routine presets certain default values. The defaults include:

- o **Modes** variable set to zero—this means you select a low-resolution display.
- o **Next** variable set to zero—no other **ViewPort** is linked to this one. If you want to have multiple **ViewPorts** in a single **View**, you must create the link yourself. The last **ViewPort** in the chain must have a **Next** value of 0.

If you have defined two **ViewPorts**, such as

```
struct ViewPort vpA;  
struct ViewPort vpB;
```

and you want them to both be part of the same display, you must create a link between them, and a NULL link at the end of the chain of **ViewPorts**:

```
vpA.Next = &vpB; /* tell first one the address of the second */  
vpB.Next = NULL; /* after this one, there are no others */
```

Preparing the BitMap Structure

The **BitMap** structure tells the system where to find the display and drawing memory and how this memory space is organized. The following code section prepares a **BitMap** structure, including allocation of memory for the bit-map. For this example, this memory is used only for the display and is not shared with any drawing routines. The example writes directly to the display area.

```
/* initialize the BitMap structure */  
InitBitMap( &b, DEPTH, WIDTH, HEIGHT );  
/* now allocate some memory that can be  
* be linked into the BitMap for display purposes */  
for( i=0; i<DEPTH, i++)  
{  
    b.Planes[i] = (PLANEPTR)AllocRaster(WIDTH, HEIGHT);  
}
```

This code allocates enough memory to handle the display area for as many bit-planes as the depth you have defined. This code segment does not include the error-checking that is present in the full example later on.

Preparing the RasInfo Structure

The **RasInfo** structure provides information to the system about the location of the **BitMap** as well as the positioning of the display area as a window against a larger drawing area. Use the following steps to prepare the **RasInfo** structure:

```

ri.BitMap = &b; /* specify address of the BitMap structure */
ri.RxOffset = 0;
ri.RyOffset = 0; /* match the upper lefthand corner of the
                  * display area with the upper left corner of
                  * the drawing area - see figure 1-16 */
ri.next = NULL; /* for a single playfield display, there
                  * is only one RasInfo structure present */

```

Preparing the ColorMap Structure

Interrupts should be used to display this **ViewPort**. When the **View** is created, Copper instructions are generated to change the current contents of each color register just before the topmost line of a **ViewPort** so that this **ViewPort**'s color registers will be used for interpreting its display.

Here are the steps normally used for initializing a **ColorMap**:

```

/* define some colors in an array of words */
UWORD colortable [] = { 0, 0xf00, 0x0f0, 0x00f }
/* allocate space and get a pointer to it */
/* 4 colors in this table (4 registers for Copper
   * to reload before this ViewPort is displayed */
vp.ColorMap = GetColorMap (4);
LoadRGB4( vp, ColorTable, 4 )

```

Note: The "4" in the name **LoadRGB4()** refers to the fact that each of the red, green, and blue values in a color table entry consists of four bits. It has nothing to do with the fact that this particular color table contains four entries, which is a result of the choice of **DEPTH = 2** for this example.

From the section called "ViewPort Color Selection," notice that you might need to specify more colors in the color map than you think. If you use a dual-playfield display (covered later in this chapter) with a depth of 1 for each of the two playfields, this means a total of four colors (two for each playfield). However, because playfield 2 uses color registers starting from number 8 on up when in dual-playfield mode, the color map must be initialized to contain at least 10 entries. That is, it must contain entries for colors 0 and 1 (for playfield 1) and color numbers 8 and 9 (for playfield 2). Space for sprite colors must be allocated as well.

Creating the Display Instructions

Now that you have initialized the system data structures, you can request that the system prepare a set of display instructions for the Copper using these structures as input data. During the one or more blank vertical lines that precede each **ViewPort**, the Copper is busy changing the characteristics of the display hardware to match the characteristics you expect for this **ViewPort**. This may include a change in display resolution, a change in the colors to be used, or other user-defined modifications to system registers.

Here is the code that creates the display instructions:

```
MakeVPort( &v, &vp );
```

In this line of code, **&v** is the address of the **View** structure and **&vp** is the address of the first **ViewPort** structure. Using these structures, the system has enough information to build the instruction stream that defines your display.

MakeVPort() creates a special set of instructions that controls the appearance of the display. If you are using animation, the graphics animation routines create a special set of instructions to control the hardware sprites and the system color registers. In addition, the advanced user can create special instructions (called user Copper instructions) to change system operations based on the position of the video beam on the screen.

All of these special instructions must be merged together before the system can use them to produce the display you have designed. This is done by the system routine **MrgCop()** (which stands for "Merge Coprocessor Instructions"). Here is a typical call:

```
MrgCop (&v); /* merge this View's Copper instructions  
            * into a single instruction list */
```

LOADING AND DISPLAYING THE VIEW

To display the **View**, you need to load it using **LoadView()** and turn on the direct memory access (DMA). A typical call is shown below.

```
LoadView( &v );
```

where **&v** is the address of the **View** structure defined in the example above.

Two macros control display DMA: **ON_DISPLAY** and **OFF_DISPLAY**. They simply turn the display DMA control bit in the DMA control register on or off. After you have loaded a new **View**, you use **ON_DISPLAY** to allow the system DMA to display it on the screen.

If you are drawing to the display area and do not want the user to see intermediate steps in the drawing, you can turn off the display. Because **OFF_DISPLAY** shuts down the display DMA and possibly speeds up other system operations, it can be used to provide additional memory cycles to the blitter or the 68000. The distribution of system DMA, however, allows four-channel sound, disk read/write, and a sixteen-color, low-resolution display (or four-color, high-resolution display) to operate at the same time with no slowdown (7.1 megahertz effective rate) in the operation of the 68000.

GRAPHICS EXAMPLE PROGRAM

The program below creates and displays a single-playfield display that is 320 pixels wide, 200 lines high, and two bit-planes deep.

```
#include "exec/types.h"
#include "graphics/gfx.h"
#include "hardware/dmabits.h"
#include "hardware/custom.h"
#include "hardware/blit.h"
#include "graphics/gfxmacros.h"
#include "graphics/copper.h"
#include "graphics/view.h"
#include "graphics/gels.h"
#include "graphics/regions.h"
#include "graphics/clip.h"
#include "exec/exec.h"
#include "graphics/text.h"
#include "graphics/gfxbase.h"

#define DEPTH 2
#define WIDTH 320
#define HEIGHT 200
#define NOT_ENOUGH_MEMORY -1000
/* construct a simple display */

struct View v;
struct ViewPort vp;
struct ColorMap *cm;      /* pointer to ColorMap structure, dynamic alloc */
struct RasInfo ri;
struct BitMap b;
struct RastPort rp;

LONG i;
SHORT j,k,n;
```

```

extern struct ColorMap *GetColorMap();
struct GfxBase *GfxBase;

struct View *oldview;      /* save pointer to old View so can restore */

                          /* black, red, green, blue */
USHORT colortable[] = { 0x000, 0xf00, 0x0f0, 0x00f }; /* my own colors */
SHORT boxoffsets[] = { 802, 2010, 3218 };           /* where to draw boxes */

UBYTE *displaymem;
UWORD *colorpalette;

main()
{
    GfxBase = (struct GfxBase *)OpenLibrary("graphics.library",0);
    if (GfxBase == NULL) exit(1);
    oldview = GfxBase->ActiView; /* save current View to restore later */
    /* example steals screen from Intuition if Intuition is around */

    InitView(&v);           /* initialize View */
    InitVPort(&vp);        /* init ViewPort */
    v.ViewPort = &vp;     /* link View into ViewPort */

    /* init bit map (for RasInfo and RastPort) */
    InitBitMap(&b,DEPTH,WIDTH,HEIGHT);

    /* (init RasInfo) */
    ri.BitMap = &b;
    ri.RxOffset = 0;
    ri.RyOffset = 0;
    ri.Next = NULL;

    /* now specify critical characteristics */
    vp.DWidth = WIDTH;
    vp.DHeight = HEIGHT;
    vp.RasInfo = &ri;

    /* (init color table) */
    cm = GetColorMap(4);           /* 4 entries, since only 2 planes deep */
    colorpalette = (UWORD *)cm->ColorTable;
    for(i=0; i<4; i++) {
        *colorpalette++ = colortable[i];
    }
    /* copy my colors into this data structure */
    vp.ColorMap = cm;           /* link it with the ViewPort */
}

```

```

/* allocate space for bitmap */
for(i=0; i<DEPTH; i++)
{
    b.Planes[i] = (PLANEPTR)AllocRaster(WIDTH,HEIGHT);
    if(b.Planes[i] == NULL) exit(NOT_ENOUGH_MEMORY);
}

MakeVPort( &v, &vp );    /* construct Copper instruction (prelim) list */
MrgCop( &v );            /* merge preliminary lists together into a real
                          * Copper list in the view structure. */

for(i=0; i<2; i++)
{
    displaymem = (UBYTE *)b.Planes[i];
    BltClear(displaymem,RASSIZE(WIDTH,HEIGHT),0)
}

LoadView(&v);
/* now fill some boxes so that user can see something */
/* always draw into both planes to assure true colors */
for(n=1; n<4; n++)      /* three boxes */
{
    for(k=0; k<2; k++)
    {
        /* boxes will be in red, green and blue */
        displaymem = b.Planes[k] + boxoffsets[n-1];
        DrawFilledBox(n,k);
    }
}

Delay(50*10);          /* wait for 10 seconds */

LoadView(oldview);    /* put back the old View */

FreeMemory();         /* exit gracefully */
CloseLibrary(GfxBase); /* since program opened library, close it */

} /* end of main() */

/* return user- and system-allocated memory to sys manager */
FreeMemory()
{
    /* free drawing area */
    for(i=0; i<DEPTH; i++)

```

```

    {
        FreeRaster(b.Planes[i],WIDTH,HEIGHT);
    }
    /* free the color map created by GetColorMap() */
    FreeColorMap(cm);
    /* free dynamically created structures */
    FreeVPortCopLists(&vp);
    FreeCprList(v.LOFCprList);
    return(0);
}
DrawFilledBox(fillcolor,plane)
SHORT fillcolor,plane;
{
    UBYTE value;
    for(j=0; j<100; j++)
    {
        if(((fillcolor & (1 << plane)) != 0)
        {
            value = 0xff;
        }
        else
        {
            value = 0;
        }
        for(i=0; i<20; i++)
        {
            *displaymem++ = value;
        }
        displaymem += (b.BytesPerRow - 20);
    }
    return(0);
}

```

Exiting Gracefully

The sample program above provides a way of exiting gracefully, returning to the memory manager all dynamically-allocated memory chunks. Notice the calls to **FreeRaster()** and **FreeColorMap()**. These calls correspond directly to the allocation calls **AllocRaster()** and **GetColorMap()** located in the body of the program. Now look at the calls within **FreeMemory()** to **FreeVPortCopLists()** and **FreeCprList()**. When you call **MakeVPort()**, the graphics system dynamically allocates some space to hold intermediate instructions from which a final Copper instruction list is created. When you call **MrgCop()**, these intermediate Copper lists are merged together into the final Copper list, which is then given to the hardware

for interpretation. It is this list that provides the stable display on the screen, split into separate **ViewPorts** with their own colors and resolutions and so on.

When your program completes, you must see that it returns all of the memory resources that it used so that those memory areas are again available to the system for reassignment to other projects. Therefore, if you use the routines **MakeVPort()** or **MrgCop()**, you must also arrange to use **FreeCprList()** (pointing to each of those lists in the **View** structure) and **FreeVPortCopLists()** (pointing to the **ViewPort** that is about to be deallocated). If your view is interlaced, you will also have to call **FreeCprList(&v.SHFCprList)** because an interlaced view has a separate Copper list for each of the two fields displayed.

As a final caveat, notice that when you do free everything, the memory manager or other programs may immediately change the contents of the freed memory. Therefore, if the Copper is still executing an instruction stream (as a result of a previous **LoadView()**) when you free that memory, the display will go "south." You will probably want to turn off the display or provide an alternate Copper list when this one is to be deallocated.

Advanced Topics

CREATING A DUAL-PLAYFIELD DISPLAY

In dual-playfield mode, you have two separately controllable playfields. In this mode, you always define two **RasInfo** data structures. Each of these structures defines one of the playfields. There are seven different ways you can configure a dual-playfield display, because there are five different distributions of the bit-planes which the system hardware allows. Table 1-4 shows these distributions.

Table 1-4: Bit-Plane Assignment in Dual-playfield Mode

Number of Bit-planes	Playfield 1 Depth	Playfield 2 Depth
0	0	0
1	1	0
2	1	1
3	2	1
4	2	2
5	3	2
6	3	3

Recall that if you set PFBA in the **ViewPort Modes** variable to 1, you can swap playfield priority and display playfield 2 in front of playfield 1. In this way, you can get more bit-planes in the background playfield than you have in the foreground playfield. If you create a display with multiple **ViewPorts**, only for this **ViewPort** will the playfield priority be changed.

Playfield 1 is defined by the first of the two **RasInfo** structures. Playfield 2 is defined by the second of the two **RasInfo** structures.

When you call **MakeVPort()**, you use parameters as follows:

```
MakeVPort( &view, &viewport );
```

The **ViewPort Modes** variable must include the DUALPF bit. This tells the graphics system that there are two **RasInfo** structures to be used.

In summary, to create a dual-playfield display you must do the following things:

- o Allocate one **View** structure
- o Allocate two **BitMap** structures
- o Allocate two **RasInfo** structures (linked together), each pointing to different **BitMaps**
- o Allocate one **ViewPort** structure
- o Set up a pointer in the **ViewPort** structure to the playfield 1 **RasInfo**

- Initialize each **BitMap** structure to describe one playfield, using one of the permissible bit-plane distributions shown in table 1-4 and allocate memory for the bit-planes themselves. Note that **BitMap 1** and **BitMap 2** need *not* be the same width and height.
- Initialize the **ViewPort** structure
- Set the DUALPF (and possibly the PFBA) bit in the **ViewPort Modes** variable
- Call **MakeVPort()**
- Call **MrgCop()**

For display purposes, each of the two **BitMaps** is assigned to a separate playfield display.

To draw separately into the **BitMaps**, you must also assign these **BitMaps** to two separate **RastPorts**. The section called “Initializing the RastPort” shows you how to use a **RastPort** data structure to control your drawing routines.

CREATING A DOUBLE-BUFFERED DISPLAY

To produce smooth animation or other such effects, it is occasionally necessary to double-buffer your display. To prevent the user from seeing your graphics rendering while it is in progress, you will want to draw into one memory area while actually displaying a different area.

Double-buffering consists of creating two separate display areas and two sets of pointers to those areas for a single **View**.

To create a double-buffered display, you must perform these actions:

- Allocate two **BitMap** structures
- Allocate one **RasInfo** structure
- Allocate one **ViewPort** structure
- Allocate one **View** structure
- Initialize each **BitMap** structure to describe one drawing area and allocate memory for the bit-planes themselves
- Create a pointer for each **BitMap**

- o Create a pointer for the **View** long-frame Copper list (**LOFCprList**) and short-frame Copper list (**SHFCprList**) for each of two alternate display fields. The **SHFCprList** is for interlaced displays.
- o Initialize the **RasInfo** structure, setting the **BitMap** pointer to point to one of the two **BitMaps** you have created
- o Call **MakeVPort()**
- o Call **MrgCop()**
- o Call **LoadView()**

When you call **MrgCop()**, the system uses all of the information you have provided in the various data structures to create a list of instructions for the Copper to execute. This list tells the Copper how to split the display and how to specify colors for the various portions of the display. When the steps shown above have been completed, the system will have allocated memory for a long-frame (LOF) Copper list and a short-frame (SHF) Copper list and will have set pointers called **LOFCprList** and **SOFCprList** in the **View** structure. The long-frame Copper list is normally used for all non-interlaced displays, and the short-frame Copper list is used only when interlaced mode is turned on. The pointers point to the two sets of Copper instructions.

The **LOFCprList** and **SHFCprList** pointers are initialized when **MrgCop()** is called. The instruction stream referenced by these pointers includes references to the first **BitMap**.

You must now do the following:

- o Save the current values in back-up pointers and set the values of **LOFCprList** and **SHFCprlist** in the **View** structure to zero. When you next perform **MrgCop()**, the system automatically allocates another memory area to hold a new list of instructions for the Copper.
- o Install the pointer to the other **BitMap** structure in the **RasInfo** structure before your call to **MakeVPort()**, and then call **MakeVPort** and **MrgCop**.

Now you have created two sets of instruction streams for the Copper, one of which you have saved in a pair of pointer variables. The other has been newly created and is in the **View** structure. You can save this new set of pointers as well, swapping in the set that you want to use for display, while drawing into the **BitMap** that is not on the display. Remember that you will have to call **FreeCprList()** on both sets of Copper lists when you have finished.

HOLD-AND-MODIFY MODE

In hold-and-modify mode you can create a single-playfield display in which 4,096 different colors can be displayed simultaneously. This requires that your **ViewPort** be defined using six bit-planes and that you set the HAM bit in the **ViewPort Modes** variable.

When you draw into the **BitMap** associated with this **ViewPort**, you can choose one of four different ways of drawing into the **BitMap**. (Drawing into a **BitMap** is shown in the next section, "Drawing Routines.") If you draw using color numbers 0-15, the pixel you draw will appear in the color specified in that particular system color register. If you draw with any other color value from 16-31, the color displayed depends on the color of the pixel that is to the immediate left of this pixel on the screen. For example, hold constant the contents of the red and the green parts of the previously produced color, and take the rest of the bits of this new pixel's color register number as the new contents for the blue part of the color. Hold-and-modify means hold part and modify part of the preceding defined pixel's color.

Note that a particular hold-and-modify pixel can only change one of the three color values at a time. Thus, the effect has a limited control.

In hold-and-modify mode, you use all six bit-planes. Planes 5 and 6 are used to modify the way bits from planes 1 - 4 are treated, as follows:

- o If the 6-5 bit combination from planes 6 and 5 for any given pixel is 00, normal color selection procedure is followed. Thus, the bit combinations from planes 4 - 1, in that order of significance, are used to choose one of 16 color registers (registers 0-15).
If only five bit-planes are used, the data from the sixth plane is automatically supplied with the value as 0.
- o If the 6-5 bit combination is 01, the color of the pixel immediately to the left of this pixel is duplicated and then modified. The bit combinations from planes 4 - 1 are used to replace the four "blue" bits in the pixel color without changing the value in any color register.
- o If the 6-5 bit combination is 10, the color of the pixel immediately to the left of this pixel is duplicated and then modified. The bit combinations from planes 4 - 1 are used to replace the four "red" bits.
- o If the 6-5 bit combination is 11, the color of the pixel immediately to the left of this pixel is duplicated and then modified. The bit combinations from planes 4 - 1 are used to replace the four "green" bits.
- o At the leftmost edge of each line, hold-and-modify begins with the background color. The color choice does *not* carry over from the preceding line.

Drawing Routines

Most of the graphics drawing routines require information about how the drawing is to take place. For this reason, the graphics support routines provide a data structure called a **RastPort**, which contains information essential to the graphics drawing functions. In using most of the drawing functions, you must pass them a pointer to your **RastPort** structure. Associated with the **RastPort** is another data structure called a **BitMap**, which contains a description of the organization of the data in the drawing area.

INITIALIZING A BITMAP STRUCTURE

The **RastPort** contains information for controlling the drawing. In order to use the graphics, you also need to tell the system the memory area location where the drawing will occur. You do this by initializing a **BitMap** structure, defining the characteristics of the drawing area, as shown in the following example. This was already shown in the section called "Forming a Basic Display," but it is repeated here because it relates to drawing as well as to display routines. You need not necessarily use the same **BitMap** for both the drawing and the display.

```
struct BitMap myBitMap;
SHORT depth = 3; /* max of eight colors ... going to need three
                  * bit-planes to represent this number of colors */
SHORT width = 320;
SHORT height = 200;

InitBitMap( &myBitMap, depth, width, height);
```

INITIALIZING A RASTPORT STRUCTURE

Before you can use a **RastPort** for drawing, you must initialize it. Here is a sample initialization sequence:

```
struct RastPort myRastPort;
InitRastPort(&myRastPort);

/* now link together the BitMap and the RastPort */
myRastPort.BitMap = &myBitMap;
```

Note that you cannot perform the link until after the **RastPort** has been initialized.

The **RastPort** data structure can be found in the include files *rastport.h* and *rastport.i*. It contains the following information:

- o Drawing pens
- o Drawing modes
- o Patterns
- o Text attributes and font information
- o Area-filling information
- o Graphics elements information for animation
- o Current pen position
- o A write mask
- o Some graphics private data
- o A pointer for user extensions

The following sections explain each of the items in the **RastPort** structure.

Drawing Pens

The Amiga has three different drawing “pens” associated with the graphics drawing routines. These are:

- o **FgPen**—the foreground or primary drawing pen. For historical reasons, it is also called the A-Pen.
- o **BgPen**—the background or secondary drawing pen. For historical reasons, it is also called the B-Pen.
- o **AOIPen**—the area outline pen. For historical reasons, it is also called the O-Pen.

A drawing pen variable in the **RastPort** contains the current value (range 0-255) for a particular color choice. This value represents a color register number whose contents are to be used in rendering a particular type of image. In essence, the bits of a “pen” determine which bit-planes are affected when a color is written into a pixel (as determined by the drawing mode and modified by the pattern variables and the write mask as described below). The drawing routines support **BitMaps** up to eight planes deep, allowing for future expansion in the hardware.

Note: The Amiga 1000 contains only 32 color registers. Any range beyond that repeats the colors in 0-31. For example, pen numbers 32-63 refer to the colors in registers 0-31.

The color in **FgPen** is used as the primary drawing color for rendering lines and areas. This pen is used when the drawing mode is JAM1 (see the next section for drawing modes). JAM1 specifies that only one color is to be “jammed” into the drawing area.

You establish the color for **FgPen** using the statement:

```
SetAPen( &myRastPort, newcolor );
```

The color in **BgPen** is used as the secondary drawing color for rendering lines and areas. If you specify that the drawing mode is JAM2 (jamming two colors) and a pattern is being drawn, the primary drawing color (**FgPen**) is used where there are 1s in the pattern. The secondary drawing color (**BgPen**) is used where there are 0s in the pattern.

You establish the drawing color for **BgPen** using the statement:

```
SetBPen( &myRastPort, newcolor );
```

The area outline pen **AOIPen** is used in two applications: area fill and flood fill. (See “Area Fill Operations” below.) In area fill, you can specify that an area, once filled, can be outlined in this **AOIPen** color. In flood fill (in one of its operating modes) you can fill until the flood-filler hits a pixel of the color specified in this pen variable.

You establish the drawing color for **AOIPen** using the statement:

```
SetOPen( &myRastPort, newcolor );
```

Drawing Modes

Four drawing modes may be specified:

- JAM1 Whenever you execute a graphics drawing command, one color is jammed into the target drawing area. You use only the primary drawing pen color, and for each pixel drawn, you *replace* the color at that location with the **FgPen** color.

- JAM2 Whenever you execute a graphics drawing command, two colors are jammed into the target drawing area. This mode tells the system that the pattern variables (both line pattern and area pattern—see the next section) are to be used for the drawing. Wherever there is a 1 bit in the pattern variable, the **FgPen** color replaces the color of the pixel at the drawing position. Wherever there is a 0 bit in the pattern variable, the **BgPen** color is used.

COMPLEMENT

For each 1 bit in the the pattern, the corresponding bit in the target area is complemented—that is, its state is reversed. As with all other drawing modes, the write mask can be used to protect specific bit-planes from being modified. Complement mode is often used for drawing and then erasing lines.

INVERSEVID

This is the drawing mode used primarily for text. If the drawing mode is (JAM1 | INVERSEVID), the text appears as a transparent letter surrounded by the **FgPen** color. If the drawing mode is (JAM2|INVERSEVID), the text appears as in (JAM1|INVERSEVID) except that the **BgPen** color is used to draw the text character itself. In this mode, the roles of **FgPen** and **BgPen** are effectively reversed.

You set the drawing modes using the statement:

```
SetDrMd( &myRastPort, newmode );
```

Patterns

The **RastPort** data structure provides two different pattern variables that it uses during the various drawing functions: a line pattern and an area pattern. The line pattern is 16 bits wide and is applied to all lines. When you initialize a **RastPort**, this line pattern value is set to all 1s (hex FFFF), so that solid lines are drawn. You can also set this pattern to other values to draw dotted lines if you wish. For example, you can establish a dotted line pattern with the statement:

```
SetDrPt( &myRastPort, 0xcccc );
```

where “cccc” is a bit-pattern, 1100110011001100, to be applied to all lines drawn. If you draw multiple, connected lines, the pattern cleanly connects all the points.

The area pattern is 16 bits wide and its height is some power of two. This means that you can define patterns in heights of 1, 2, 4, 8, 16, and so on. To tell the system how large a pattern you are providing, include this statement:

```
SetAfPt( &myRastPort, &myAreaPattern, power_of_two );
```

where **&myAreaPattern** is the address of the first word of the area pattern and **power_of_two** specifies how many words are in the pattern. For example:

```

USHORT myAreaPattern[] = {
    0xff00,
    0xff00,
    0x00ff,
    0x00ff,
    0xf0f0,
    0xf0f0,
    0x0f0f,
    0x0f0f
};

SetAfPt( &myRastPort, &myAreaPattern, 3 );

```

This example produces a pattern that is a large checkerboard above a small checkerboard. Because **power_of_two** is set to 3, the pattern is 2 to the 3rd, or 8, rows high.

Pattern Positioning

The pattern is always positioned with respect to the upper left corner of the **RastPort** drawing area (the 0,0 coordinate). If you draw two rectangles whose edges are adjacent, the pattern will be continuous across the rectangle boundaries.

Multicolored Patterns

The last example above produces a two-color pattern with one color where there are 1s and the other color where there are 0s in the pattern. A special mode allows you to develop a pattern having up to 256 colors. To create this effect, specify **power_of_two** as a negative value instead of a positive value.

The following initialization establishes an 8-color checkerboard pattern where each square in the checkerboard has a different color. The checkerboard is 2 squares wide by 4 squares high.

```

USHORT myAreaPattern[3][8] = {
    {
        0x0000,    /* plane 0 pattern */
        0x0000,
        0xffff,
        0xffff,
        0x0000,
        0x0000,
        0xffff,
        0xffff,
    },

```

```

    {
        0x0000,    /* plane 1 pattern */
        0x0000,
        0x0000,
        0x0000,
        0xffff,
        0xffff,
        0xffff,
        0xffff,
    },
    {
        0xff00,    /* plane 2 pattern */
        0xff00,
        0xff00,
        0xff00,
        0xff00,
        0xff00,
        0xff00,
        0xff00,
        0xff00
    }
};

SetAfPt( &myRastPort, &myAreaPattern, -3 );

/* when doing this, it is best to set three other parameters as follows: */
SetAPen( &myRastPort, 255);
SetBPen( &myRastPort, 0);
SetDrMd( &myRastPort, JAM2);

```

If you use this multicolored pattern mode, you must provide as many planes of pattern data as there are planes in your **BitMap**.

Text Attributes

Text attributes and font information are set by calls to the font routines. These are covered separately in chapter 4, "Text."

Area-fill Information

Two structures in the **RastPort**—**AreaInfo** and **TmpRas**—define certain information for area filling operations. The **AreaInfo** pointer is initialized by a call to the routine **InitArea()**.

```
InitArea (&myRastPort, &areabuffer, count);
```

To use area fill, you must first provide a work space in memory for the system to store the list of points that define your area. You must allow a storage space of 5 bytes per vertex. To create the areas in the work space, you use the functions **AreaMove()**, **AreaDraw()**, and **AreaEnd()**.

Typically, you prepare the **RastPort** for area-filling using a sequence like the following:

```
UWORD areabuffer [250];
    /* allow up to 100 vertices in the definition of an area */
InitArea (&myRastPort, &areabuffer[0], 100);
```

The area buffer *must* start on a *word* boundary. That is why the sample declaration shows **areabuffer** as composed of unsigned words (250), rather than unsigned bytes (500). It still reserves the same amount of space, but aligns the data space correctly.

In addition to the **AreaInfo** structure in the **RastPort**, you must also provide the system with some work space to build the object whose vertices you are going to define. This requires that you initialize a **TmpRas** structure, then point to that structure for your **RastPort** to use.

Here is sample code that builds and initializes a **TmpRas**. Note that the area to which **TmpRas.RasPtr** points must be at least as large as the area (width times height) of the largest rectangular region you plan to fill. Typically, you allocate a space as large as a single bit-plane (usually 320 by 200 bits for low-resolution mode, 640 by 200 bits for high-resolution mode).

```
PLANEPTR myplane;
myplane = AllocRaster(320,200);      /* get some space */
if (myplane == 0) exit(1);          /* stop if no space */
myRastPort.TmpRas = InitTmpRas(&myTmpRas,
    myplane,RASSIZE(320,200));
```

When you use functions that dynamically allocate memory from the system, you must remember to return these memory blocks to the system before your program exits. See the description of **FreeRaster()** in the “Library Summaries” appendix.

Graphics Element Pointer

The graphics element pointer in the **RastPort** structure is called **GelsInfo**. If you are doing graphics animation using the GELS system, this pointer must refer to a properly initialized **GelsInfo** structure. See chapter 3, "Animation," for more information.

Current Pen Position

The graphics drawing routines keep the current position of the drawing pen in the variables **cp_x** and **cp_y**, for the horizontal and vertical positions, respectively. The coordinate location 0,0 is in the upper left corner of the drawing area. The x value increases proceeding to the right; the y value increases proceeding toward the bottom of the drawing area.

Write Mask

The write mask is a **RastPort** variable that determines which of the bit-planes are currently writable. For most applications, this variable contains all 1s (hex ff). This means that all bit-planes defined in the **BitMap** are affected by a graphics writing operation. You can selectively disable one or more bit-planes by simply specifying a 0 bit in that specific position in the control byte. For example:

```
myRastPort.Mask = 0xFB;    /* disable bit-plane 2 */
```

USING THE GRAPHICS DRAWING ROUTINES

This section shows you how to use the Amiga drawing routines. All of these routines work either on their own or with the windowing system and layer library. See chapter 2, "Layers," or *Intuition: The Amiga User Interface* for details about using the layer library and windows.

As you read this section, keep in mind that to use the drawing routines, you need to pass them a pointer to a **RastPort**. You can define the **RastPort** directly, as shown in the sample program segments in preceding sections, or you can get a **RastPort** from your **Window** structure using code like the following:

```
struct Window *w;  
struct RastPort *usableRastPort;  
    /* and then, after your Window is initialized... */  
usableRastPort = w->RastPort;
```

You can also get the **RastPort** from the layer structure, if you are not using Intuition.

Drawing Individual Pixels

You can set a specific pixel to a desired color by using a statement like this:

```
int result;  
result = WritePixel( &myRastPort, x, y);
```

WritePixel() uses the primary drawing pen and changes the pixel at that x,y position to the desired color if the x,y coordinate falls within the boundaries of the **RastPort**. A value of 0 is returned if the write was successful; a value of -1 is returned if x,y was outside the range of the **RastPort**.

Reading Individual Pixels

You can determine the color of a specific pixel with a statement like this:

```
int result;  
result = ReadPixel( &myRastPort, x, y);
```

ReadPixel() returns the value of the pixel color selector (from 0 to 255) at the specified x,y location. If you specify an x,y outside the range of your **RastPort**, this function returns a value of -1.

Drawing Lines

Two functions are associated with line drawing: **Move()** and **Draw()**. **Move()** simply moves the cursor to a new position. It is like picking up a drawing pen and placing it at a new location. This function is executed by the statement:

```
Move( &myRastPort, x, y);
```

Draw() draws a line from the current x,y position to a new x,y position specified in the statement itself. The drawing pen is left at the new position. This is done by the statement:

```
Draw( &myRastPort, x, y);
```

Draw() uses the pen color specified for **FgPen**. Here is a sample sequence that draws a red line from location (0,0) to (100,50). Assume that the value in color register 2 represents red.

```
SetAPen( &myRastPort, 2);      /* make primary pen red */
Move( &myRastPort, 0, 0);      /* move to new location */
Draw( &myRastPort, 100,50);    /* draw to a new location */
```

Caution: If you attempt to draw a line outside the bounds of the **BitMap**, using the basic initialized **RastPort**, you may crash the system. You must either do your own software clipping to assure that the line is in range, or use the layer library. Software clipping means that you need to determine if the line will fall outside your **BitMap** *before* you draw it.

Drawing Patterned Lines

To turn the example above into a patterned line draw, simply add the following statement:

```
SetDrPt( &myRastPort, 0xaaaa);
```

Now all lines drawn appear as dotted lines. To resume drawing solid lines, execute the statement:

```
SetDrPt( &myRastPort, -1);
```

Drawing Multiple Lines with a Single Command

You can use multiple **Draw()** statements to draw connected line figures. If the shapes are all definable as interconnected, continuous lines, you can use a simpler function, called **PolyDraw()**. **PolyDraw()** takes a set of line endpoints and draws a shape using these points. You call **PolyDraw()** with the statement:

```
PolyDraw( &myRastPort, count, arraypointer);
```

PolyDraw() reads an array of points and draws a line from the current pen position to the first, then a connecting line to each succeeding position in the array until **count** points have been drawn. This function uses the current drawing mode, pens, line pattern, and write mask specified in the target **RastPort**; for example:

```

SHORT linearray[ ] = {
    3,3,
    15,3,
    15,15,
    3,15,
    3,3
};
PolyDraw( &myRastPort, 5, &linearray[0]);

```

draws a rectangle, using the five defined pairs of x,y coordinates.

Area-fill Operations

Assuming that you have properly initialized your **RastPort** structure to include a properly initialized **AreaInfo**, you can perform area fill by using the functions described in this section.

AreaMove() tells the system to begin a new polygon, closing off any other polygon that may already be in process by connecting the end-point of the previous polygon to its starting point. **AreaMove()** is executed with the statement:

```
AreaMove( &myRastPort, x, y);
```

AreaDraw() tells the system to add a new vertex to a list that it is building. No drawing takes place when **AreaDraw()** is executed. It is executed with the statement:

```
AreaDraw( &myRastPort, x, y);
```

AreaEnd() tells the system to draw all of the defined shapes and fill them. When this function is executed, it obeys the drawing mode and uses the line pattern and area pattern specified in your **RastPort** to render the objects you have defined. Note that to fill an area, you do not have to **AreaDraw()** back to the first point before calling **AreaEnd()**. **AreaEnd()** automatically closes the polygon. **AreaEnd()** is executed with the following statement:

```
AreaEnd( &myRastPort);
```

Here is a sample program segment that includes the **AreaInfo** initialization. It draws a pair of disconnected triangles, using the currently defined **FgPen**, **BgPen**, **AOIPen**, **DrawMode**, **LinePtrn**, and **AreaPtrn**:

```

WORD areabuffer[250];
struct RastPort *rp;

```



```

struct TmpRas tmpr;
struct AreaInfo myAreaInfo;

InitArea(&myAreaInfo, areabuffer, 100);
rp->AreaInfo = &myAreaInfo;
rp->TmpRas = InitTmpRas( &tmpr, AllocRaster(320,200), RASSIZE(320,200));

/* Area routines need a temporary raster buffer at least as large as the
 * largest object to be drawn. If a single task uses multiple RastPorts,
 * it is sometimes possible to share the same TmpRas structure among
 * multiple RastPorts. Multiple tasks, however, cannot share a TmpRas,
 * as each task won't know when another task has a drawing partially
 * completed.
 */

AreaMove( rp, 0,0 );
AreaDraw( rp, 0,100);
AreaDraw( rp, 100,100);

AreaMove( rp, 50,10);
AreaDraw( rp, 50,50);
AreaDraw( rp, 100,50);

AreaEnd ( rp );

```

If you had executed the statement “SetOPen(&myRastPort, 3)” in the area-fill example, then the areas that you had defined would have been outlined in pen color 3. To turn off the outline function, you have to set the **RastPort Flags** variable back to 0 by:

```

#include "graphics/gfxmacros.h"

BNDRYOFF(&myRastPort);

```

Otherwise, every subsequent area-fill or rectangle-fill operation will use the outline pen.

Caution: If you attempt to fill an area outside the bounds of the **BitMap**, using the basic initialized **RastPort**, it may crash the system. You must either do your own software clipping to assure that the area is in range, or use the layer library.

Flood-fill Operations

Flood fill is a technique for filling an arbitrary shape with a color. The Amiga flood-fill routines can use a plain color or do the fill using a combination of the drawing mode, **FgPen**, **BgPen**, and the area pattern.

There are two different modes for flood fill:

- o In *outline mode* you specify an x,y coordinate, and from that point the system searches outward in all directions for a pixel whose color is the same as that specified in the area outline pen. All horizontally or vertically adjacent pixels *not* of that color are filled with a colored pattern or plain color. The fill stops at the outline color. Outline mode is selected when the **mode** variable is a 0.
- o In *color mode* you specify an x,y coordinate, and whatever pixel color is found at that position defines the area to be filled. The system searches for all horizontally or vertically adjacent pixels whose color is the same as this one and replaces them with the colored pattern or plain color. Color mode is selected when the **mode** variable is a 1.

You use the **Flood()** routine for flood fill. The syntax for this routine follows.

```
Flood( rp, mode, x, y);
```

where

- rp** is a pointer to the **RastPort**
- x,y** is the starting coordinate in the **BitMap**
- mode** tells how to do the fill

The following sample program fragment creates and then flood-fills a triangular region. The overall effect is exactly the same as shown in the preceding area-fill example above, except that flood-fill is slightly slower than area-fill. Mode 0 (fill to a pixel that has the color of the outline pen) is used in the example.

```

oldAPen = myRastPort.FgPen;
SetAPen( &myRastPort, myRastPort.AOIPen);
/* using mode 0 */
/* triangular shape */
Move( &myRastPort, 0, 0);
Draw( &myRastPort, 0, 100);
Draw( &myRastPort, 100, 100);
Draw( &myRastPort, 0, 0); /* close it */

SetAPen( &myRastPort, oldAPen);
Flood(&myRastPort, 0, 10, 50);

```

This example saves the current **FgPen** value and draws the shape in the same color as **AOIPen**. Then **FgPen** is restored to its original color so that **FgPen**, **BgPen**, **DrawMode**, and **AreaPtrn** can be used to define the fill within the outline.

Rectangle-fill Operations

The final fill function, **RectFill()**, is for filling rectangular areas. The form of this function follows:

```
RectFill( rp, xmin, ymin, xmax, ymax);
```

where

xmin and **ymin**

represent the upper left corner of the rectangle

xmax and **ymax**

represent the lower right corner of the rectangle

rp points to the **RastPort** that receives the filled rectangle

Rectangle-fill uses **FgPen**, **BgPen**, **AOIPen**, **DrawMode** and **AreaPtrn** to fill the area you specify. Remember that the fill can be multicolored as well as single- or two-colored.

The following three sets of statements perform exactly the same function:

```

/* area-fill a rectangular area */
SetAPen(rp,1);
SetOPen(rp,3);
AreaMove(rp,0,0);
AreaDraw(rp,0,100);
AreaDraw(rp,100,100);
AreaDraw(rp,100,0);
AreaEnd(rp);

/* flood-fill a rectangular area */
SetAPen(rp,3);
SetOPen(rp,3);
Move(rp,0,0);
Draw(rp,0,100);
Draw(rp,100,100);
Draw(rp,100,0);
Draw(rp,0,0);
SetAPen(rp,1);
Flood(rp,0,50,50);

/* rectangle-fill a rectangular area */
SetAPen(rp,1);
SetOPen(rp,3);
Rectfill(rp,0,0,100,100);

```

Not only is the `RectFill()` routine the shortest, it is also the fastest to execute.

Data Move Operations

The graphics support functions include several routines for simplifying the handling of the rectangularly organized data that you would encounter when doing raster-based graphics. These routines do the following:

- o Clear an entire segment of memory
- o Set a raster to a specific color
- o Scroll a subrectangle of a raster
- o Draw a pattern “through a stencil”
- o Extract a pattern from a bit-packed array and draw it into a raster
- o Copy rectangular regions from one bit-map to another
- o Control and utilize the hardware-based data mover, the blitter

The following sections cover these routines in detail.

Clearing a Memory Area

For memory that is accessible to the blitter (that is, internal CHIP memory), the most efficient way to clear a range of memory is to use the blitter. You use the blitter to clear a block of memory with the statement:

```
BltClear( memblock, bytecount, flags);
```

where **memblock** is a pointer to the location of the first byte to be cleared, and **bytecount** is the number of bytes to set to zero.

This command accepts the starting location and count and clears that block to zeros. For the meanings of settings of the **flags** variable, see the summary page for this routine in the “Library Summaries” appendix.

Setting a Whole Raster to a Color

You can preset a whole raster to a single color by using the function **SetRast()**. A call to this function takes the following form:

```
SetRast( RastPort, pen);
```

where

RastPort

is a pointer to the **RastPort** you wish to use

pen

is the pen value that you wish to fill that **RastPort**

Scrolling a Sub-rectangle of a Raster

You can scroll a sub-rectangle of a raster in any direction—up, down, left, right, or diagonally. To perform a scroll, you use the **ScrollRaster()** routine and specify a dx and dy (delta-x, delta-y) by which the rectangle image should be moved towards the (0,0) location.

As a result of this operation, the data within the rectangle will become physically smaller by the size of delta-x and delta-y, and the area vacated by the data when it has been cropped and moved is filled with the background color (color in **BgPen**).

Here is the syntax of the **ScrollRaster()** function:

```
ScrollRaster( rp, dx, dy, xmin, ymin, xmax, ymax );
```

where

rp is a pointer to a **RastPort**

dx, dy

are the distances (positive, 0, or negative) to move the rectangle

xmin, xmax, ymin, ymax

specify the outer bounds of the sub-rectangle

Here are some examples that scroll a sub-rectangle:

```
/* scroll down 2 */  
ScrollRaster(&myRastPort,0,2,10,10,50,50);
```

```
/* scroll right 1 */  
ScrollRaster(&myRastPort,1,0,10,10,50,50);
```

Drawing through a Stencil

The routine **BltPattern()** allows you to change only a very selective portion of a drawing area. Basically, this routine lets you define the rectangular region to be affected by this drawing operation and a mask of the same size that defines how that area will be affected.

Figure 1-17 shows an example of what you can do with **BltPattern()**. The 0 bits are represented by blank rectangles; the 1 bits by filled-in rectangles.

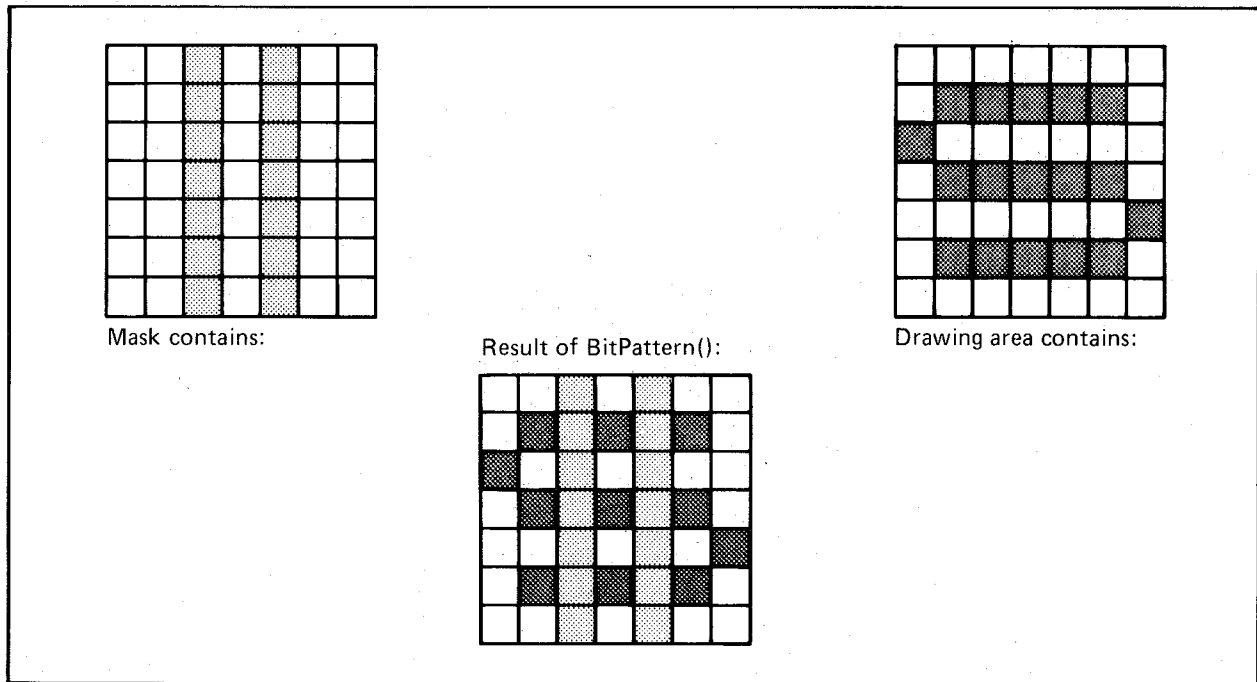


Figure 1-17: Example of Drawing Through a Stencil

In the “Result” drawing, the lighter squares show where the target drawing area has been affected. Exactly *what* goes into the drawing area where the mask has 1’s is determined by your **FgPen**, **BgPen**, **DrawMode**, and **AreaPtrn**.

The variables that control this function are:

- rastport** a pointer to the drawing area
- mask** a pointer to the mask (mask layout explained below)
- xl, maxx** upper left corner x, and lower right corner x

yl, maxy upper left corner y, and lower right corner y

bytecnt number of bytes per row for the mask (*must* be an even number of bytes)

You call **BltPattern()** with:

BltPattern(rastport, mask, xl, yl, maxx, maxy, bytecnt)

The **mask** parameter is a rectangularly organized, contiguously stored pattern. This means that the pattern is stored in linearly increasing memory locations stored as (**maxy - yl**) rows of **bytecnt** bytes per row.

Note: These patterns must obey the same rules as **BitMaps**. This means that they must consist of an even number of bytes per row. For example, a mask such as:

```
01000010000000000
00100100000000000
00011000000000000
00100100000000000
```

is stored in memory beginning at a legal *word* address.

Extracting from a Bit-packed Array

You use the routine **BltTemplate()** to extract a rectangular area from a source area and place it into a destination area. Figure 1-18 shows an example.

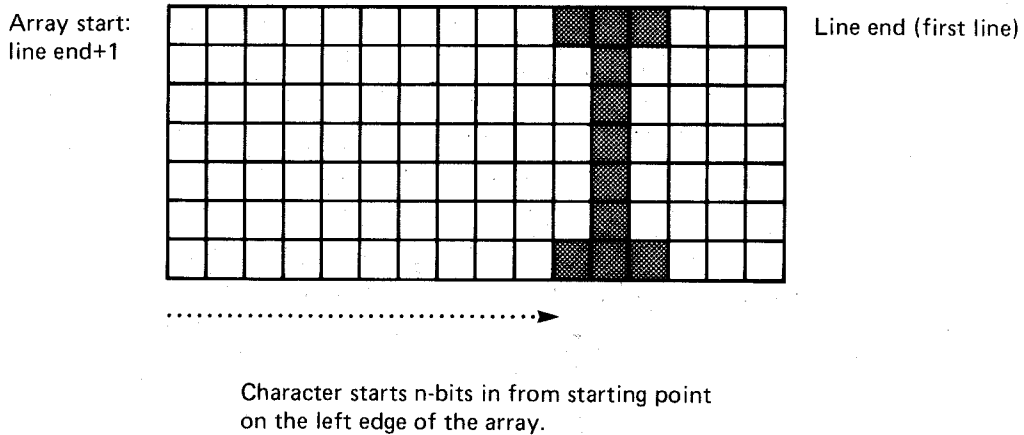


Figure 1-18: Example of Extracting from a Bit-Packed Array

If the rectangular bit array is to be represented as a rectangle within a larger, rectangularly organized bit array, the system must know how the larger array is organized. This allows the system to extract each line of the object properly. For this extraction to occur properly, you need to tell the system the modulo for the array. The modulo is the value that must be added to the address pointer so that it points to the correct word in the next line in this rectangularly organized array.

Figure 1-19 represents a single bit-plane and the smaller rectangle to be extracted. The modulo in this instance is 4, because at the end of each line, you must add 4 to the address pointer to make it point to the first word in the smaller rectangle.

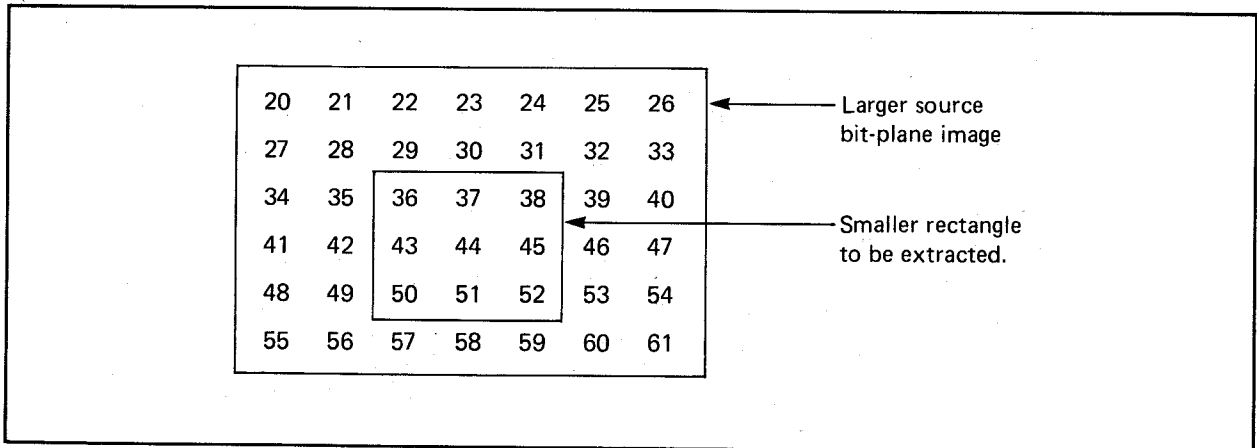


Figure 1-19: Modulo

Note that the modulo value must be an even number of bytes.

BltTemplate() takes the following arguments:

source	the source pointer for the array
srcX	source X (bit position) in the array at which the rectangle begins
srcMod	source modulo so it can find the next part of the source rectangle
destRastPort	the destination RastPort
destX, destY	destination x and y, showing where to put the rectangle
sizeX, sizeY	size x and y, indicating how much data to move

You call **BltTemplate()** with:

```
BltTemplate( source, srcX, srcMod, destRastPort, destX, destY, sizeX, sizeY );
```

BltTemplate() uses **FgPen**, **BgPen**, **DrawMode** and **Mask** to place the template into the destination area. This routine differs from **BltPattern()** in that only a solid color is deposited in the destination drawing area, with or without a second solid color as the background (as in the case of text). Also, the template can be arbitrarily bit-aligned and sized in x.

Copying Rectangular Areas

Two routines copy rectangular areas from one section of chip memory to another: **BltBitMap()** and **ClipBlit()**. **BltBitMap()** is the basic routine, taking **BitMaps** as part of its arguments. It allows you to define a rectangle in a source region and copy it to a destination area of the same size elsewhere in memory. This routine is often used in graphics rendering.

ClipBlit() takes most of the same arguments, but it works with the **RastPorts** and layers. Before **ClipBlit()** moves data, it looks at the area from which and to which the data is being copied (**RastPorts**, not **BitMaps**) and determines if there are overlapping areas involved. It then splits up the overall operation into a number of bit maps to move the data in the way you request.

Here is a sample call to **ClipBlit()**. This call is used in an image editor to transfer a rectangular block of data from the screen to a back-up area.

```

ClipBlit( &rastport, /* on-screen area */
          x,y,        /* upper left corner of rectangle */
          &undorastport, /* screen editor can undo things, has
                        * a RastPort specifically for undo */
          0,0,        /* upper left corner of destination */
          SIZEx,SIZEy /* how big is the rectangle */
          minterm);

```

The **minterm** variable is an unsigned byte value whose leftmost 4 bits represent the action to be performed during the move. This routine uses the blitter device to move the data and can therefore logically combine or change the data as the move is made. The most common operation is a direct copy from source area to destination, which is the hex value C0.

You can determine how to set the **minterm** variable by using the logic equations shown in table 1-5.

Table 1-5: Minterm Logic Equations

Logic Term in Leftmost 4 Bits	Logic Term Included in Final Output
8	BC
4	\overline{BC}
2	$\overline{B}C$
1	$\overline{B}\overline{C}$

Source B contains the data from the source rectangle, and source C contains the data from the destination area. If you choose bits 8 and 4 from the logic terms (C0), in the final destination area you will have data that occurs in source B only. Thus, C0 means a direct copy. The logic equation for this is:

$$BC + \overline{BC} = B(C + \overline{C}) = B$$

Logic equations may be used to decide on a number of different ways of moving the data. For your convenience, a few of the most common ones are listed in table 1-6.

Table 1-6: Some Common Logic Equations for Copying

Hex Value	Mode
30	Replace destination area with inverted source B.
50	Replace destination area with inverted version of original of destination.
60	Put B where C is not, put C where B is not (cookie cut).
80	Only put bits into destination where there is a bit in the same position for both source and destination (sieve operation).

Refer to the listing for **BltBitMap()** in the “Library Summaries” index.

Accessing the Blitter in a Multitasking Environment

To use the blitter, you must first be familiar with how its registers control its operation. This topic is covered thoroughly in the *Amiga Hardware Reference Manual* and is not repeated here.

Four routines may be used to gain access to the blitter:

- o **OwnBlitter()** allows your task to obtain exclusive use of the blitter. Note, however, that the system uses the blitter extensively for disk and display operation. While your task is using the blitter, many other system processes will be locked out. Therefore, use it only for brief periods and relinquish it as quickly as possible, using **DisownBlitter()**.
- o **DisownBlitter()** returns the device to shared operation.
- o **QBlit()** and **QBSBlit()** let your task queue up requests for the use of the blitter on a non-exclusive basis. You share the blitter with system tasks.

You provide a data structure called a **bltnode** (blitter node). The system can use this structure to link blitter usage requests into a first-in, first-out (FIFO) queue. When your turn comes, your own blitter routine can be repeatedly called until your routine says it is finished using the blitter.

Two separate queues are formed. One queue is for the **QBlit()** routine. You use **QBlit()** when you simply want something done and you do not necessarily care when it happens. This may be the case when you are moving data in a memory area that is not currently being displayed.

The second queue is maintained for **QBSBlit()**. QBS stands for “queue-beam-synchronized” blitter operations. **QBSBlit()** forms a beam-synchronized FIFO. When the video beam gets to a predetermined position, your routine is called. Beam synchronization takes precedence over the simple FIFO. This means that if the beam sync matches, the beam-synchronous blit will be done before the non-synchronous blit in the first position in the queue. You might use **QBSBlit()** to draw into an area of memory that is currently being displayed to modify memory that has already been “passed-over” by the video beam. This avoids display flicker as an area is being updated.

The input to each routine is a pointer to a **bltnode** data structure. The required items of the data structure are:

- o A pointer to a **bltnode**
- o A pointer to a function to perform
- o A **beamsync** value (used if this is a **beamsync** blit)
- o A status flag indicating whether the blitter control should perform a “clean-up” routine when the last blit is finished
- o The address of the clean-up routine if the status flag states that it should be used

The **bltnode** data structure is contained in the include file *hardware/blit.h*. Here is a copy of that data structure, followed by details about the items you must initialize:

```
struct bltnode
{
    struct    bltnode *n;
    int      (*function)( );
    char     stat;
    short    blitsize;
    short    beamsync;
    int      (*cleanup)( );
};
```

The contents of **bltnode** are as follows:

```
struct bltnode *n;
```

This is a pointer to the next **bltnode**, which, for most applications will be zero. You should not link **bltnodes** together. This is to be performed by the system by way of a separate call to **QBlit()** or **QBSBlit()**.

int (*function)();

This position is occupied by the address of a function that the blitter queuer will call when your turn comes up. Your routine must be formed as a subroutine, with an **RTS** at the end. Using the C-language convention, the returned value will be in D0 (C returns its value by the **return(value)** statement).

If you return a nonzero value, the system will call your routine the next time the blitter is done until you finally return 0. This is to allow you to maintain control over the blitter; for example, it allows you to handle all five bit-planes if you are blitting an object that spans that number of planes. For display purposes, if you are blitting multiple objects and then saving and restoring the background, you must be sure that all planes of the object are positioned before another object is overlaid. This is the reason for the lockup in the blitter queue; it allows all work per object to be completed before going on to the next one.

Actually, the system tests the *status codes* for a condition of **EQUAL** or **NOTEQUAL**. When the C language returns the value of 0, it sets the status codes to **EQUAL**. When it returns a value of -1, it sets the status codes to **NOTEQUAL**, so they would be compatible. Functions **(*function)()** that are written for **QBlit()** and **QBSBlit()** are not normally written in C. They are usually written in assembly language, as they then can take advantage of the ability of the queue routines to pass them parameters in the system registers. The register passing conventions for these routines are as follows:

- o Register A0 receives a pointer to the system hardware registers so that all hardware registers can be referenced as an offset from that address.
- o Register A1 contains a pointer to the current **bltnode**. You may have queued up multiple blits, each of which perhaps uses the same blitter routine. You can access the data for this particular operation as an offset from the value in A1. A typical user of these routines will precalculate the hardware register values that are stuffed into the registers and, during the routine, simply stuff them. For example, you can create a new structure such as the following:

```

struct myblit {
    struct bltnode; /* make this new structure
                    * compatible with the bltnode
                    * by making it the first element */
    short bltcon1; /* contents to be stuffed into
                  * blitter control register 1 */
    short fwmask,lwmask;
                    /* first and last word masks */
    short bltmde, bltmdb, bltmda;
                    /* modulus for sources a, b, and c */
    char *bltpta, *bltptb, *bltptc;
                    /* pointer to source data for sources */
};

```

Other forms of data structures are certainly possible, but this should give you the general idea.

char stat;

Tells the system whether or not to execute the clean-up routine at the end. This byte should be set to CLEANUP (0x40) if cleanup is to be performed. If not, then the **bltnode cleanup** variable can be zero.

short beamsync;

The value that should be in the VBEAM counter for use during a beam-synchronous blit before the **function()** is called.

The system cooperates with you in planning when to start a blit in the routine **QBSBlit()** by not calling your routine until, for example, the video beam has already passed by the area on the screen into which you are writing. This is especially useful during single buffering of your displays. There may be time enough to write the object between scans of the video display. You will not be visibly writing while the beam is trying to scan the object. This avoids flicker (part of an old view of an object along with part of a new view of the object).

int (*cleanup)();

The address of a routine that is to be called after your last return from the **QBlit()** routine. When you finally return a zero, the queuer will call this subroutine (ends in **RTS** or **return()**) as the clean-up. Your first entry to the function may have dynamically allocated some memory or may have done something that must be undone to make for a clean exit. This routine must be specified.

User Copper Lists

The Copper coprocessor allows you to produce mid-screen changes in certain hardware registers in addition to changes that the system software already provides. For example, it is the Copper that allows the Amiga to split the viewing area into multiple draggable screens, each with its own independent set of colors.

To create your own mid-screen (or mid-Intuition-Screen) effects on the system hardware registers, you provide “user Copper lists” that can be merged into the system Copper lists.

In the **ViewPort** data structure there is a pointer named **UCopIns**. If this pointer value is non-NULL, it points to a user Copper list that you have dynamically allocated and initialized to contain your own special hardware-stuffing instructions. You allocate a user Copper list by an instruction sequence such as the following:

```
struct UCopList *cl;

cl = (struct UCopList *)
    AllocMem(sizeof(struct UCopList), MEMF_PUBLIC |
             MEMF_CHIP | MEMF_CLEAR);
```

Once this pointer to a user Copper list is available, you can use it with system macros (*graphics/gfxmacros.h*) to instruct the system what to add to its own list of things for the Copper to do within a specific **ViewPort**.

The file *graphics/gfxmacros.h* provides the following three macro functions that implement user Copper instructions.

CWAIT waits for the video beam to reach a particular horizontal and vertical position. Its format follows:

```
CWAIT(uc, v, h)
```

where

uc is the pointer to the Copper list

v is the vertical position for which to wait, specified relative to the top of the **ViewPort**. The legal range of values is from 0 to 261.

h is the horizontal position for which to wait. The legal range of values is from 0 to 223

CMOVE installs a particular value into a specified system register. Its format follows:

CMOVE(uc, reg, value)

where

uc is the pointer to the Copper list

reg is the register to be affected, specified in this form form: "custom.register" (see *hardware/custom.h*)

CEND terminates the user Copper list. Its format follows:

CEND(uc)

where **uc** is the pointer to the user Copper list.

Executing any of the user Copper list macros causes the system to dynamically allocate special data structures called intermediate Copper lists that are linked into your user Copper list (the list to which **cl** points) describing the operation. When you call the function **MakeVPort(&view, &viewport)** as shown in the section called "Forming A Basic Display," the system uses all of its intermediate Copper lists to sort and merge together the real Copper lists for the system (**LOFCprList** and **SHFCprList**).

When your program exits, you must return to the system all of the memory that you allocated or caused to be allocated. This means that you must return the intermediate Copper lists, as well as the user Copper list data structure. Here are two different methods for returning this memory to the system.

```
/* Returning memory to the system if you have NOT
 * obtained the viewport from Intuition. */
```

```
FreeVPortCopLists(&viewport);
```

```
/* Returning memory to the system if you HAVE
 * obtained the viewport from Intuition. */
```

```
CloseScreen(screen); /* Intuition only */
```

The example program below shows the use of user Copper lists under Intuition.

```
/* User-Copper-Lists Demo Program ... changes the background color
 * in mid-screen.
 */
```

```
#define WINDOWGADGETS (WINDOWSIZING|WINDOWDRAG|
    WINDOWDEPTH|WINDOWCLOSE)
```

```
#define WWIDTH 120
```

```
#define WHEIGHT 90
```

```
#define MAXINT 0xFFFFFFFF
```

```
#include "exec/types.h"
```

```
#include "exec/memory.h"
```

```
#include <graphics/gfxmacros.h>
```

```
#include <graphics/copper.h>
```

```
#include "intuition/intuition.h"
```

```
#include <hardware/custom.h>
```

```
extern struct Window *OpenWindow();
```

```
extern struct Screen *OpenScreen();
```

```
long IntuitionBase=0;
```

```
long GfxBase=0;
```

```
/* use the 40/80 column font for this test */
```

```
struct TextAttr TestFont = {
```

```
    "topaz.font", 8, 0, 0
```

```
};
```

```
struct NewScreen ns = {
```

```
    0, 0,          /* start position */
```

```
    320, 200, 4,   /* width, height, depth */
```

```
    0, 1,          /* detail pen, block pen */
```

```
    0,             /* viewing mode */
```

```
    CUSTOMSCREEN, /* screen type */
```

```
    &TestFont,     /* font to use */
```

```
    "Test Screen", /* default title for screen */
```

```
    NULL           /* pointer to additional gadgets */
```

```
};
```

```
extern struct Custom custom;
```

```
/* provides a way to get to the base of the custom chips */
```

```

main()
{
    struct Window *w;      /* pointer to a Window */
    struct RastPort *rp;   /* pointer to a RastPort */
    struct ViewPort *vp;  /* pointer to a ViewPort */
    struct UCopList *cl;  /* user Copper list and a pointer to it. */
    struct Screen *screen;
    GfxBase = OpenLibrary("graphics.library", 0);
    if (GfxBase == NULL)
    {
        exit(1000);
    }
    IntuitionBase = OpenLibrary("intuition.library", 0);
    if (IntuitionBase == NULL)
    {
        CloseLibrary(GfxBase);
        exit(2000);
    }
    screen = OpenScreen(&ns);
    if(!screen)
    {
        goto cleanup;
    }
    else
    {
        vp = &screen->ViewPort;
        rp = &screen->RastPort;
    }

    /* v1.1 initialization, just use CINIT for v1.2 */

    /* In this case, although WE allocated the memory for the user Copper list,
     * the SYSTEM (Intuition) deallocates it when the custom screen is closed.
     * Therefore there is no corresponding FreeMem() in this sample program.
     */
    cl = AllocMem(sizeof(struct UCopList),MEMF_PUBLIC|MEMF_CLEAR);

    CWAIT(cl,100,0);          /* wait till middle of screen */
    CMOVE(cl,custom.color[0],0xFFFF); /* change background color */
    CEND(cl);

    /* Programmer can affect ANY of the system registers that the Copper has access to
     * (see the Amiga Hardware Reference Manual) in this way. Simply note that the
     * system may already be using these registers in some manner and that most of
     * the system registers are either read-only or write-only, so you'll have to be

```

```

    * careful about what you are trying to affect.
    */
    vp->UCopIns = cl;

    Delay(50);    /* wait one second before changing anything */

    /* Now force a remake of the Copper list for all screens. */

    RethinkDisplay();

    Delay(100);
    CloseScreen(screen);
cleanup:
    CloseLibrary(IntuitionBase);
    CloseLibrary(GfxBase);
}

/* end of main() */

```

Advanced Graphics Examples

DUAL-PLAYFIELDS EXAMPLE

This example is almost identical to the single-playfield demonstration program earlier in this chapter. It has been adapted to show a dual-playfield display with objects drawn in both playfields. The single playfield wrote directly into the screen's memory. This example adds a **RastPort** so that rectangle-fill routines can be used.

```

#include <exec/types.h>
#include <graphics/gfx.h>
#include <graphics/gfxbase.h>
#include <hardware/dmabits.h>
#include <hardware/custom.h>
#include <graphics/gfxmacros.h>
#include <graphics/rastport.h>
#include <graphics/view.h>
#include <exec/exec.h>

#define DEPTH 2
#define WIDTH 320
#define HEIGHT 200
#define NOT_ENOUGH_MEMORY -1000

```

```

struct View v;
struct ViewPort vp;
struct ColorMap *cm; /* pointer to ColorMap structure, dynamic alloc */
struct RasInfo ri;
struct BitMap b;

/* added a second RasInfo for dual.playfield */
struct RasInfo ri2;
/* added a second BitMap for dual.playfield */
struct BitMap b2;

short i,j,k,n;
struct ColorMap *GetColorMap();
struct GfxBase *GfxBase;

    /* black, red, green, blue,
    * ignored, ignored, ignored, ignored,
    * (transparent), purple, lime green, mauve */
USHORT colortable[] = {
    0x000, 0xf00, 0x0f0, 0x00f,
    0,0,0,0,
    0,    0x495, 0x62a, 0xf9c
};
/* Nobody will see center set of 4 colors in this case because only two planes
* and dual-playfield mode. (In dualpf mode, colors 0-7 are dedicated to
* playfield 1, and 8-15 to playfield number 2. So since only 2 planes in each
* playfield, colors 4-7 and 12-15 won't even get used in this example)
*/

UWORD *colorpalette;

/* added RastPorts for both bitmaps */

struct RastPort rp, rp2;
struct View *oldview; /* save and restore old View */

main()
{
    GfxBase = (struct GfxBase *)OpenLibrary("graphics.library",0);
    if (GfxBase == NULL) exit(1);

    InitView(&v); /* initialize View */
    v.ViewPort = &vp; /* link View into ViewPort */
    InitVPort(&vp); /* init ViewPort */

```

```

                /* now specify critical characteristics */
vp.DWidth = WIDTH;
vp.DHeight = HEIGHT;
vp.RasInfo = &ri;
vp.Modes = DUALPF | PFBA ; /* dual-playfield mode */

/* init bit map (for RasInfo and RastPort) */
InitBitMap(&b,DEPTH,WIDTH,HEIGHT);
    /* (init RasInfo) */
ri.BitMap = &b;
/* align upper left corners of display
 * with upper left corner of drawing area */
ri.RxOffset = 0;
ri.RyOffset = 0;

/* ***** */
/* changed here for dual playfields */
    InitBitMap(&b2,DEPTH,WIDTH,HEIGHT);
    ri.Next = &ri2;
    ri2.BitMap = &b2;
    ri2.RxOffset = 0;
    ri2.RyOffset = 0;
    ri2.Next = 0;
/* ***** */

                /* (init color table) */
cm = GetColorMap(12); /* 12 entries, since dual playfields */
colorpalette = cm->ColorTable;
for(i=0; i<12; i++)
{
    *colorpalette++ = colortable[i];
}

                /* copy my colors into this data structure */
vp.ColorMap = cm; /* link it with the ViewPort */

                /* allocate space for BitMap */
for(i=0; i<DEPTH; i++)
{
    b.Planes[i] = (PLANEPTR)AllocRaster(WIDTH,HEIGHT);
    if(b.Planes[i] == NULL) exit(NOT_ENOUGH_MEMORY);
    b2.Planes[i] = (PLANEPTR)AllocRaster(WIDTH,HEIGHT);
    if(b2.Planes[i] == NULL) exit(NOT_ENOUGH_MEMORY);
}

/* Initialize the RastPorts and link them to the bitmaps */

```

```

InitRastPort(&rp);
InitRastPort(&rp2);
rp.BitMap = &b;
rp2.BitMap = &b2;

MakeVPort( &v, &vp );    /* construct Copper instr (prelim) list */
MrgCop( &v );           /* merge prelim lists together into a real
                        * Copper list in the View structure. */
SetRast(&rp,0);         /* simpler form of setting drawing area to 0 */
SetRast(&rp2,0);

oldview = GfxBase->ActiView; /* save current view to restore later */
/* example steals screen from Intuition if started from WBench */

LoadView(&v);

/* Now fill some boxes so that user can see something */
/* first playfield */
SetAPen(&rp,1);
RectFill(&rp,20,20,200,100);
SetAPen(&rp,2);
RectFill(&rp,40,40,220,120);
SetAPen(&rp,3);
RectFill(&rp,60,60,240,140);
/* second playfield */
SetAPen(&rp2,1);
RectFill(&rp2,50,90,245,180);
SetAPen(&rp2,2);
RectFill(&rp2,70,70,265,160);
SetAPen(&rp2,3);
RectFill(&rp2,90,10,285,148);

/* Now tear some holes in the playfield so user can see that foreground
 * area of playfield 2 (called PFB also) is transparent in any area
 * where it has a color value of 0
 */

SetAPen(&rp2,0);
RectFill(&rp2,110,15,130,175);
RectFill(&rp2,175,15,200,175);
Delay(300);    /* uses AmigaDOS function... delay 5 seconds */
LoadView(oldview); /* Put Intuition's View back again */
WaitTOF();      /* wait for Intuition View to return */
FreeMemory();  /* and exit gracefully */
CloseLibrary(GfxBase);

```

```

} /* end of main() */

FreeMemory()
{
    /* return user and system-allocated memory to sys manager */

    for(i=0; i<DEPTH; i++) /* free the drawing area */
    {
        FreeRaster(b.Planes[i],WIDTH,HEIGHT);
        FreeRaster(b2.Planes[i],WIDTH,HEIGHT);
    }
    FreeColorMap(cm); /* free the color map */
    /* free dynamically created structures */
    FreeVPortCopLists(&vp);
    FreeCprList(v.LOFCprList);
    return(0);
}

```

HOLD-AND-MODIFY MODE EXAMPLE

This example demonstrates the Amiga's hold-and-modify mode, showing at all times a different subset of 256 of the 4,096 colors available on the Amiga. At any moment, no two squares are the same color.

```

/*****
 * Rob Peck -- November 5, 1985
 * Bob Pariseau -- November 10, 1985 (Rework for tutorial)
 *****/

#include <exec/types.h>
#include <intuition/intuition.h>
#include <intuition/intuitionbase.h>

#define XSIZE 11 /* Color box sizes */
#define YSIZE 6

struct GfxBase *GfxBase; /* Export the library pointers */
struct IntuitionBase *IntuitionBase;

struct RastPort *rp; /* Graphics structures */
struct ViewPort *vp;

struct TextAttr TestFont =
{
    "topaz.font", /* Standard system font */

```



```

    8, 0, 0
};

struct Window    *w; /* Intuition structures */
struct Screen    *screen;
struct IntuiMessage *message;

struct NewScreen ns = {
    0, 0, /* start position */
    320, 200, 6, /* width, height, depth */
    0, 1, /* detail pen, block pen */
    HAM, /* Hold and Modify ViewMode */
    CUSTOMSCREEN, /* screen type */
    &TestFont, /* font to use */
    " 256 different out of 4096", /* default title for screen */
    NULL /* pointer to additional gadgets */
};

struct NewWindow nw = {
    0, 11, /* start position */
    320, 186, /* width, height */
    -1, -1, /* detail pen, block pen */
    MOUSEBUTTONS|CLOSEWINDOW, /* IDCMP flags */
    ACTIVATE|WINDOWCLOSE, /* window flags */
    NULL, /* pointer to first user gadget */
    NULL, /* pointer to user checkmark */
    "colors at any given moment", /* window title */
    NULL, /* pointer to screen (set below) */
    NULL, /* pointer to superbitmap */
    0, 0, 320, 186, /* ignored since not sizeable */
    CUSTOMSCREEN /* type of screen desired */
};

LONG squarecolor[16 * 16], freecolors[4096-(16*16)];
SHORT squares[16 * 16];
SHORT xpos[16], ypos[16];

char *number[] = {
    "0", "1", "2", "3", "4", "5", "6", "7", "8", "9",
    "A", "B", "C", "D", "E", "F"
};

SHORT sStop, cStop, sequence;
BOOL textneeded;

```

```

main()
{
    ULONG class;
    USHORT code, i;
    BOOL wheelmode;

    for(i=0; i<16; i++) /* establish color square positions */
    {
        xpos[i] = (XSIZE + 4) * i + 20;
        ypos[i] = (YSIZE + 3) * i + 21;
    }

    GfxBase = (struct GfxBase *)OpenLibrary("graphics.library", 0);
    if (GfxBase == NULL) exit(100);

    IntuitionBase = (struct IntuitionBase *)OpenLibrary("intuition.library", 0);
    if (IntuitionBase == NULL)
    {
        CloseLibrary(GfxBase);
        exit(200);
    }

    screen = (struct Screen *)OpenScreen(&ns);
    if (screen == NULL)
    {
        CloseLibrary(IntuitionBase);
        CloseLibrary(GfxBase);
        exit(300);
    }

    nw.Screen = screen; /* open window in our new screen */
    w = (struct Window *)OpenWindow(&nw);
    if (w == NULL)
    {
        CloseScreen(screen);
        CloseLibrary(IntuitionBase);
        CloseLibrary(GfxBase);
        exit(400);
    }

    vp = &screen->ViewPort; /* Set colors in screen's VP */
    rp = w->RPort; /* Render into the window's RP */

    /* Set the color registers: Black, Red, Green, Blue, White */

```

```

SetRGB4(vp, 0, 00, 00, 00);
SetRGB4(vp, 1, 15, 00, 00);
SetRGB4(vp, 2, 00, 15, 00);
SetRGB4(vp, 3, 00, 00, 15);
SetRGB4(vp, 4, 15, 15, 15);

SetBPen(rp, 0);    /* Insure clean text */
textneeded = TRUE;
wheelmode = TRUE; /* Start with Color Wheel display */

for (i,j) {
{ /* Process any and all messages in the queue, then update the display
   * colors once, then come back here to look at the queue again. If you
   * see a left-mouse-button-down event, then switch display modes. If you
   * see a Close-Window-gadget event, then clean up and exit the program.
   * NOTE: This is a BUSY LOOP so the colors will cycle as quickly as possible.
   */

while((message = (struct IntuiMessage *)GetMsg(w->UserPort)) != NULL)
{
    class = message->Class;
    code = message->Code;
    ReplyMsg(message); /* Can't reply until done using it! */

    if(class == CLOSEWINDOW) /* Exit the program */
    {
        CloseWindow(w);
        CloseScreen(screen);
        CloseLibrary(IntuitionBase);
        CloseLibrary(GfxBase);
        exit(0);
    }

    if(class == MOUSEBUTTONS && code == SELECTDOWN) /* swap modes */
    {
        wheelmode = NOT wheelmode;

        SetAPen(rp, 0); /* Clear the drawing area */
        SetDrMd(rp, JAM1);
        RectFill(rp, 3, 12, 318, 183);
        textneeded = TRUE;
    }
}
if(wheelmode) colorWheel(); else colorFull();
}

```

```
}
```

```
colorFull() /* Display a randomized set of colors */
```

```
{
```

```
    SHORT sChoice, cChoice, usesquare;
```

```
    LONG usecolor;
```

```
    if(textneeded) /* First call since mode change? */
```

```
    {
```

```
        prompt();
```

```
        sStop = 255; /* Top of list of squares yet to change */
```

```
        cStop = 4095 - 256; /* Top of list of colors still needing use */
```

```
        for(usecolor=0; usecolor<256; usecolor++) /* Initialize colors */
```

```
        {
```

```
            usesquare = usecolor;
```

```
            squares[usesquare] = usesquare;
```

```
            squarecolor[usesquare] = usecolor;
```

```
            hamBox(usecolor, xpos[usesquare % 16], ypos[usesquare / 16]);
```

```
        }
```

```
        for(usecolor=256; usecolor<4095; usecolor++) /* Ones not yet used */
```

```
        {
```

```
            freecolors[usecolor - 256] = usecolor;
```

```
        }
```

```
    }
```

```
/******
```

```
* Randomly choose next square to change such that all squares change color
```

```
* at least once before any square changes twice. squares[0] through squares
```

```
* [sStop] are the square numbers that have not yet changed in this pass.
```

```
* RangeRand(r) is an integer function provided in "amiga.lib" that produces
```

```
* a random result in the range 0 to (r-1) given an integer r in the range 1 to 65535.
```

```
*****/
```

```
sChoice = RangeRand(sStop + 1); /* Pick a remaining square */
```

```
usesquare = squares[sChoice]; /* Extract square number */
```

```
squares[sChoice] = squares[sStop]; /* Swap it with sStop slot */
```

```
squares[sStop] = usesquare;
```

```
if(NOT sStop--) sStop = 255; /* Only one change per pass */
```

```
/******
```

```

* Randomly choose new color for selected square such that all colors are
* used once before any color is used again, and such that no two squares
* simultaneously have the same color. freecolors[0] through freecolors[cStop]
* are the colors that have not yet been chosen in this pass. Note that
* the 256 colors in use at the end of the previous pass are not available
* for choice in this pass.

```

```

*****/

```

```

cChoice = RangeRand(cStop + 1);

```

```

usecolor = freecolors[cChoice];
freecolors[cChoice] = freecolors[cStop];
freecolors[cStop] = squarecolor[usesquare];
squarecolor[usesquare] = usecolor;

```

```

if(NOT cStop-- cStop = 4095 - 256;

```

```

    hamBox(usecolor, xpos[usesquare % 16], ypos[usesquare / 16]);

```

```

}

```

```

colorWheel() /* Display an ordered set of colors */

```

```

{

```

```

    SHORT i, j;

```

```

    if(textneeded)

```

```

    {

```

```

        prompt();

```

```

        SetAPen(rp, 2); /* Green pen for green color numbers */

```

```

        Move(rp, 260, ypos[15]+17);

```

```

        Text(rp, "Green", 5);

```

```

        for(i=0; i<16; i++)

```

```

        {

```

```

            Move(rp, xpos[i]+3, ypos[15]+17);

```

```

            Text(rp, number[i], 1);

```

```

        }

```

```

        SetAPen(rp, 3); /* Blue pen for blue color numbers */

```

```

        Move(rp, 4, 18);

```

```

        Text(rp, "Blue", 4);

```

```

        for(i=0; i<16; i++)

```

```

        {

```

```

            Move(rp, 7, ypos[i]+6);

```

```

            Text(rp, number[i], 1);

```

```

        }

```

```

    SetAPen(rp, 1); /* Red pen for red color numbers */
    Move(rp, 271, 100);
    Text(rp, "Red", 3);

    sequence = 0;
}

SetAPen(rp, 1); /* Identify the red color in use */
SetDrMd(rp, JAM2);
Move(rp, 280, 115);
Text(rp, number[sequence], 1);

for(j=0; j<16; j++) /* Update all of the squares */
    for(i=0; i<16; i++)
        hamBox((sequence<<8 | i<<4 | j), xpos[i], ypos[j]);

if(++sequence == 16) sequence=0;
}

prompt() /* Display mode changing prompt */
{
    SetDrMd(rp, JAM2);
    SetAPen(rp, 4);
    Move(rp, 23, 183);
    Text(rp, "[left mouse button = new mode]", 30);
    textneeded = FALSE;
}

/*****
* hamBox() -- routine to draw a colored box in Hold and Modify mode. Draws a
* box of size XSIZE by YSIZE with an upper left corner at (x,y). The
* desired color is achieved in 3 steps on each horizontal line of the box.
* First we set the red component, then the green, then the blue. We
* achieve this by drawing a vertical line of Modify-Red, followed by a
* vertical line of Modify-Green, followed by a rectangle of Modify-Blue.
* Note that the resulting color for the first two vertical lines depends
* upon the color(s) of the pixels immediately to the left of that
* line. By the time we reach the rectangle we are assured of getting
* (and maintaining) the desired color because we have set all 3
* components (R, G, and B) straight from the bit map.
*****/
hamBox(color, x, y)
LONG color, x, y;
{

```

```

SHORT c;

SetDrMd(rp, JAM1);    /* Establish Drawing Mode in RastPort */

c=((color & 0xf00)>>8); /* Extract desired Red color component. */
SetAPen(rp, c + 0x20); /* Hold G, B from previous pixel. Set R=n. */
Move(rp, x, y);
Draw(rp, x, y+YSIZE);

x++;
c=((color & 0xf0)>>4); /* Extract desired Green color component. */
SetAPen(rp, c + 0x30); /* Hold R, B from previous pixel. Set G=n. */
Move(rp, x, y);
Draw(rp, x, y+YSIZE);

x++;
c=(color & 0xf);      /* Extract desired Blue color component.*/
SetAPen(rp, c + 0x10); /* Hold R, G from previous pixel. Set B=n. */
RectFill(rp, x, y, x+XSIZE-2, y+YSIZE);
}

```

Chapter 2

Layers

The layers library enables you to create displays containing overlapping display elements. This chapter describes the layers library routines and how you use them in creating graphics.

Introduction

The layers library contains routines that do the following:

- o Multiplex a **BitMap** among various tasks by creating “layers” in the **BitMap**
- o Create separate writable **BitMap** areas, some portions of which may be in the common (perhaps on-screen) **BitMap**, and some portions in an obscured area. In two modes, called smart-refresh and superbitmap, graphics are rendered into both the obscured and the non-obscured areas.
- o Move, size or depth-arrange the layers, bringing obscured segments into a non-obscured area

Tasks can create layers in a common **BitMap** and then output graphics to those layers without any knowledge that there are other tasks currently using this **BitMap**.

To see what the layers library provides, you need only look at the Intuition user interface, as used by numerous applications on the Amiga. The windows that Intuition creates are based, in part, on the underlying strata of the layers library. You can find more details about Intuition in the book titled *Intuition: The Amiga User Interface*.

If you wish, you can use the layers library directly to create your own windowing system. The layers library takes care of the difficult things, that is, the bookkeeping jobs that are needed to keep track of where to put which bits. Once a layer is created, it may be moved, sized, depth-arranged or deleted using the routines provided in this library. In performing their rendering operations, the graphics routines know how to use the layers and only draw into the correct drawing areas.

DEFINITION OF LAYERS

The internal definition of the layers resembles a set of *clipping rectangles* in that a drawing area is split into a set of rectangles. A clipping rectangle is a rectangular area into which the graphics routines will draw. Some of the rectangles are visible; some are invisible. If a rectangle is visible, the graphics can draw directly into it. If a rectangle is obscured by an overlapping layer, the graphics routine may possibly draw into some other memory area. This memory area must be at least large enough to hold the obscured rectangle so the graphics routines can, on command, expose the obscured area.

The layers library manages interactions between the various layers by using a data structure called **Layer_Info**. Each major drawing area, called a **BitMap** (which all windows share), requires one **Layer_Info** data structure.

You may choose to split the viewing area into multiple parts by providing multiple independent **ViewPorts**. If you use the layers library to subdivide each of these parts into layers (effectively providing windows within these subdivisions), you must provide one **Layer_Info** structure for each of these parts.

TYPES OF LAYERS SUPPORTED

The layers library supports four types of layers:

- o *Simple Refresh*

No back-up area is provided. Instead, when an obscured section of the layer is exposed to view, the routine using this layer is told that a “refresh” of that area is in order. This means that the program using this layer must redraw those portions of its display that are contained in the previously obscured section of the layer. All graphics rendering routines are “clipped” so that they will only draw into exposed sections of the layer.

- o *Smart Refresh*

The system provides one or more back-up areas into which the graphics routines can draw whenever a part of this layer is obscured.

- o *Superbitmap*

There is a single back-up area, which is permanently provided to store what is not in the layer. The back-up area may be larger than the area that is actually shown in the on-screen **BitMap**.

- o *Backdrop*

A backdrop layer always appears behind all other layers that you create. The current implementation of backdrop layers prevents them from being moved, sized, or depth-arranged.

Layers Library Routines

The layers library contains the routines shown below:

Purpose	Routine
Allocating a Layer_Info structure	NewLayerInfo()
Deallocating a Layer_Info structure	DisposeLayerInfo()
Intertask operations	LockLayer(), UnLockLayer(), LockLayers(), UnlockLayers(), LockLayerInfo(), UnlockLayerInfo()
Creating and deleting layers	CreateUpfrontLayer(), CreateBehindLayer(), DeleteLayer()
Moving layers	MoveLayer()
Sizing layers	SizeLayer()
Changing a viewpoint	ScrollLayer()
Reordering layers	BehindLayer, UpfrontLayer()
Determining layer position	WhichLayer()
Sub-layer rectangle operations	SwapBitsRastPortClipRect()

INITIALIZING AND DEALLOCATING LAYERS

The function **NewLayerInfo()** allocates and initializes a **Layer_Info** data structure and allocates some extra needed memory for the 1.1 release. After the call to **NewLayerInfo()**, you can use the layer operations described in the following paragraphs.

The function **DisposeLayerInfo()** deallocates a **Layer_Info** structure that was allocated with a call to **NewLayerInfo()** and frees the extra memory that was allocated.

Note: Prior to the current 1.1 release, **Layer_Info** structures were initialized with the **InitLayers()** function. For backwards compatibility, you can still use this function with newer software. For optimal performance, however, you should call **FattenLayerInfo()** to allocate the needed extra memory and **ThinLayerInfo()** to return the memory to the system free-list. Failure to deallocate memory will result in loss of that available memory.

INTERTASK OPERATIONS

This section shows the use of the routines **LockLayerInfo()**, **UnlockLayerInfo()**, **LockLayer()**, **UnlockLayer()**, **LockLayers()**, and **UnlockLayers()**.

LockLayerInfo() and **UnlockLayerInfo()**

You create layers by using the routines **CreateUpFrontLayer()** and **CreateBehindLayer()**. If multiple tasks are all trying to create layers on the same screen or **ViewPort**, each task will be trying to affect the same data structures while creating its layers. The **Layer_Info** data structure controls the layers. **LockLayerInfo()** ensures that the **Layer_Info** data structure remains intact and tasks can obtain this exclusive access.

LockLayerInfo() grants exclusive access to the locking task. If some other task has the **Layer_Info** locked, the call will block until the lock succeeds.

LockLayer() and **Unlocklayer()**

If a task is making some changes to a particular layer, such as resizing it or moving it, the task must inhibit the graphics rendering into the layer. **LockLayer()** blocks graphics output once the current graphics function has completed. The other task goes to sleep only if it attempts to draw graphics. **LockLayer()** returns exclusive access to the layer once other tasks, including graphics, are finished with this layer.

UnlockLayer() frees the locked layer for other operations.

If more than one layer must be locked, then these **LockLayer()** calls must be surrounded by **LockLayerInfo()** and **UnLockLayerInfo()**. This is to prevent deadlock situations.

LockLayers() and **UnlockLayers()**

Sometimes it is necessary to lock all layers at the same time. For example, under Intuition, a rubber-band box is drawn when a window is being moved or sized. To draw such a box, Intuition must stop all graphics rendering to all windows (and associated layers) so that it can draw a line using the graphics complement drawing mode. If other graphics draw over this line, it would not be possible for Intuition to erase it again, using a subsequent complement operation over the same line. Thus **LockLayers()** is used to lock all layers in a single command. **UnlockLayers()** releases the layers.

You can simulate **LockLayers()** by calling **LockLayer()** for each layer in the **LayerList**. However, in that case, you must call **LockLayerInfo()** before and **UnlockLayerInfo()** after each **LockLayer()** call.

CREATING AND DELETING LAYERS

CreateUpFrontLayer() creates a layer that is in front of all other layers. Intuition uses this function to create certain types of new windows, as well as other Intuition components.

CreateBehindLayer() creates a layer that is behind all other layers. Intuition uses this function to create a new “Backdrop” window.

Each of the routines that create layers return a pointer to a layer data structure (shown in the include file *graphics/layers.h*).

Note: When you create a layer, the system automatically creates a **RastPort** to go along with it. Because a **RastPort** is specified by the drawing routines, if you use this layer’s **RastPort**, you will draw into only the area that you have designated on the screen for this layer. See also the topic called “The Layer’s RastPort” below.

DeleteLayer() is used to remove a layer from the layer list. It is one of the functions used by Intuition to close a window.

For these functions, you need to perform **LockLayerInfo()** and **UnlockLayerInfo()**, because you need to access the **Layer_Info** structure itself.

MOVING LAYERS

MoveLayer() moves a layer to a new location. When you move a layer, the move command affects the list of layers that is being managed by the **Layer_Info** data structure. The system locks the **Layer_Info** for you during this operation.

SIZING LAYERS

The **SizeLayer()** command changes the size of a layer by leaving the coordinates of the upper left corner the same and modifying the coordinates of the lower right corner of the layer. The system locks the **Layer_Info** for you during this operation.

CHANGING A VIEWPOINT

ScrollLayer() is for superbitmap layers only. This command changes the portion of a superbitmap that is shown by a layer. An analogy is a window in a wall. If the homeowner does not like the view he sees from a particular window, he might either change what he sees by planting trees (that is, new graphics rendering) or he might decide to move the window to see another part of the great outdoors (changing the portion of the superbitmap shown by a layer). You must provide a superbitmap; the **ScrollLayer()** command repositions the smaller layer against the larger superbitmap, thus showing a different part of it.

Because the layer size and on-screen position do not change while this operation is taking place, it is not necessary to lock the **Layer_Info** data structure. However, it is necessary to prevent graphics-rendering operations from drawing into this layer or its associated superbitmap while **ScrollLayer()** is performing the repositioning. Thus, the system locks the layer for you while this operation is taking place.

REORDERING LAYERS

BehindLayer() and **UpfrontLayer()** are used, respectively, to move a layer behind all other layers or in front of all other layers. **BehindLayer()** also considers any backdrop layers, moving a current layer behind all others except backdrop layers. The system performs **LockLayers()** for you during this operation.

DETERMINING LAYER POSITION

If the viewing area has been separated into several layers, you may wish to find out which layer is topmost at a particular x,y coordinate. For example, Intuition does this while keeping track of the mouse position. When you move the mouse into one of the windows and click the left button, Intuition feeds the current x,y coordinate to **WhichLayer()**. In return, **WhichLayer()** tells Intuition which layer has been selected, and thus it knows with which window you wish to work.

If you wish to be sure that no task changes the sequence of layers (by using **UpfrontLayer()**, **BehindLayer()**, **CreateUpFrontLayer()**, **DeleteLayer()**, **MoveLayer()** or **SizeLayer()**) before your task can use this information, call **LockLayerInfo()** before calling **WhichLayer()**. Then, after receiving and using the information that **WhichLayer()** delivers, you can call **UnlockLayerInfo()**. In this way, you will assure that you are acting on data that was true as of the moment it was received.

SUB-LAYER RECTANGLE OPERATIONS

The **SwapBitsClipRectRastPort()** routine is for users who do not want to worry about clipping rectangles. The need for this routine goes a bit deeper than that. It is a routine that actually enables the menu operations of Intuition to function much more quickly than they would if this routine were not provided.

Consider the case where there are several windows open on an Intuition screen. If you wish to produce a menu, there are two ways to do it:

- o Create an up-front layer with **CreateUpfrontLayer()**, then render the menu in it. This could use lots of memory and require a lot of (very temporary) “slice-and-dice” operations to create all of the clipping rectangles for the existing windows and so on.
- o Use **SwapBitsClipRectRastPort()**, directly on the screen drawing area:
 - o Render the menu in a back-up area off the screen, then lock all of the on-screen layers so that no task can use graphics routines to draw over your menu area on the screen.
 - o Next, swap the on-screen bits with the off-screen bits, making the menu appear.
 - o When you finish with the menu, swap again and unlock the layers.

The second rendering method is faster and leaves the clipping rectangles and most of the rest of the window data structures untouched.

Notice that all of the layers must be locked while the menu is visible. Any task that is using any of the layers for graphics output will be halted while the menu operations are taking place. If, on the other hand, the menu is rendered as a layer, no task need be halted while the menu is up because the lower layers need not be locked. It is a tradeoff decision that you must make.

The Layer's RastPort

When you create a layer, you automatically get a **RastPort**. The pointer to the **RastPort** is contained in the layer data structure and can be retrieved typically by the statement:

```
rp = layer->rp;          /* copy the pointer from the layer structure
                        * into a local pointer for further use */
```

Using this **RastPort**, you can draw anywhere into the layer's defined rectangle. Location (0,0) is the coordinate location for the upper left corner of the rectangle, and location (**xmax**, **ymax**) is the lower right corner. If you try to draw to any location outside of this coordinate system, the graphics routines will clip the drawing to the inside boundaries of this area.

The type of layer you specify by the **Flags** variable determines the other facilities the layer provides. The following paragraphs describe the types of layers —simple refresh, smart refresh, superbitmap, and backdrop—and the flags you set for the type you want. Note that the three layer-type **Flags** are mutually exclusive. That is, you cannot specify more than one layer-type flag—**LAYERSIMPLE**, **LAYERSMART**, **LAYERSUPER**.

SIMPLE REFRESH LAYER

When you draw into the layer, any portion of the layer that is visible (not obscured) will have its drawing rendered into the common **BitMap** of the viewing area.

If another layer operation is performed that causes part of a simple refresh layer to be obscured and then exposed, you must restore the part of the drawing that your application rendered into the obscured area.

Simple refresh has two basic advantages:

- o It uses no back-up area to save drawing sections that cannot be seen anyway (and therefore saves memory).
- o When an application tries to restore the layer by performing a full-layer redraw, (sandwiched between a **BeginUpdate()**, **EndUpdate()** pair), only those damaged areas are redrawn, making the operation very time efficient.

Its disadvantage is that the application needs to watch to see if its layer needs refreshing. This test can be performed, typically, by a statement set such as the following:

```
refreshstatus = layer->Flags & LAYERREFRESH;  
if (refreshstatus != 0) refresh(layer);
```

Note: Applications using Intuition typically get their refresh notifications as event messages passed through an Intuition Direct Communications Message Port (IDCMP).

SMART REFRESH LAYER

If any portion of the layer is hidden by another layer, the bits for that obscured portion are rendered into a back-up area. With smart refresh layers, the system handles all of the refresh requirements except when the layer is made larger. Its disadvantage is the additional memory needed to handle this automatic refresh.

SUPERBITMAP LAYER

A superbitmap layer is similar to a smart refresh layer. It too has a back-up area into which drawings are rendered for currently obscured parts of the display. However, it differs from smart refresh in that:

- o The back-up **BitMap** is user-supplied, rather than being allocated dynamically by the system.
- o The back-up **BitMap** may be larger than the area of this **BitMap** that is currently showing within the current size of this layer.

To see a larger portion of a superbitmap in the on-screen layer, you use **SizeLayer()**. To see a different portion of the superbitmap in the layer, you use **ScrollLayer()**.

When the graphics routines perform your drawing commands, part of the drawing appears in the common **BitMap** (the on-screen portion). Any drawing outside the layer itself is rendered into the superbitmap. When it is time to scroll or size the layer, the layer contents are copied into the superbitmap, the scroll or size positioning is modified, and the appropriate portions are then copied back into the layer.

BACKDROP LAYER

Any layer can be designated a backdrop layer. You can turn off the backdrop flag temporarily and allow a layer to be depth-arranged. Then by restoring the backdrop flag, you can again inhibit depth-arrangement operations.

You change the backdrop flag typically by the statements:

```
layer->Flags &= LAYERBACKDROP;    /* turn off the backdrop bit */  
layer->Flags |= LAYERBACKDROP;    /* turn on the backdrop bit */
```

Using the Layers Library

The following is a step-by-step example showing how the layers library can be used in your programs. Note that the Intuition software, which is part of the system as well, manages many of these items for you. The example below can be started up under Intuition, but it requires that the Amiga be reset in order to exit the program.

The example program explains the individual parts separately, then merges the parts into a single working example. This simple example produces three rectangles on the screen: one red, one green, and one blue. Each rectangle is rendered as a rectangle-fill of one of three smart layers created for the example.

OPENING THE LAYERS LIBRARY

Like all library routines, the layers library must be opened before it can be used. This is done typically by the following code:

```
struct LayersBase *LayersBase;
...
LayersBase = (struct LayersBase *)OpenLibrary("layers.library",0);
if(LayersBase == NULL)
{
    exit(NO_LAYERS_LIBRARY_FOUND);
}
```

OPENING THE GRAPHICS LIBRARY

Because the example uses various graphics library functions as well as the layers library, you must also open the graphics library with the following code:

```
struct GfxBase *GfxBase;
...
GfxBase = (struct GfxBase *)OpenLibrary("graphics.library",0);
if(GfxBase == NULL)
{
    exit(NO_GRAPHICS_LIBRARY_FOUND);
}
```

CREATING A VIEWING WORKSPACE

You can create a viewing workspace by using the primitives `InitVPort()`, `InitView()`, `MakeVPort()`, `MrgCop()`, and `LoadView()`. See the “Graphics Example” section in chapter 1, “Graphics Primitives.” You add the following statements:

```
struct Layer_Info *li;
li=NewLayerInfo();
```

This provides and initializes a `Layer_Info` data structure with which the system can keep track of layers that you create.

CREATING THE LAYERS

You can create layers in this common bit map by calling `CreateUpfrontLayer()` (or `CreateBehindLayer()`), with a sequence such as the following. The `Flags` value in this example is `LAYERSMART` (see `graphics/clip.h` in the “Include Files” appendix for all other flag values). This sequence requests construction of a smart refresh layer.

```
#define FLAGS LAYERSMART
struct BitMap b;
struct Layer_Info i;
struct RastPort *rp[3]; /* allocate a RastPort pointer for each layer */
struct Layer *layer[3]; /* allocate a layer pointer for each layer */

/* Layer_Info, common BitMap, x1,y1,x2,y2,
 * flags = 0 (smart refresh), null pointer to superbitmap */
layer[0] = CreateUpfrontLayer(&li,&b,20,20,100,80,FLAGS,NULL);

layer[1] = CreateUpfrontLayer(&li,&b,30,30,110,90,FLAGS,NULL);
layer[2] = CreateUpfrontLayer(&li,&b,40,40,120,100,FLAGS,NULL);

/* if not enough memory, can't continue the example */
if(layer[0]==NULL || layer[1]==NULL || layer[2]==NULL) exit(3);
```

GETTING THE POINTERS TO THE RASTPORTS

Each layer pointer data structure contains a pointer to the **RastPort** that it uses. Here is the assignment from the layer structure to a set of local pointers:

```
for(i=0; i<3; i++)
{
    rp[i] = layer[i]->rp;
}
```

USING THE RASTPORTS FOR DISPLAY

Here are the rectangle-fill operations that create the display:

```
for(i=0; i<3; i++)
{
    SetAPen(rp[i],i+1);
    SetDrMd(rp[i],JAM1);
    RectFill(rp[i],0,0,80,50);
}
```

If you perform an **UpfrontLayer()** or **BehindLayer()** command prior to the **Delay()** shown in the complete example below, all of the data contained in each layer is retained and correctly rendered automatically by the layers library. This is because these are all smart-refresh layers. If you change the example to use a **Flags** value of **LAYERSIMPLE**, and then perform **UpfrontLayer()** or **BehindLayer()**, the obscured portions of the layers, now exposed, contain only the background color. This illustrates that simple-refresh layers may have to be redrawn after layer operations are performed.

LAYERS EXAMPLE

Here is the complete example, which is a compilation of the complete example in chapter 1 and the pieces given above. Sections of the example that differ from those shown in the chapter 1 example are indicated through comments to show the additions adding the layers library demonstration.

```

/*****
 * This example shows how to use the layers.library.  Certain functions are not
 * available in the system software prior to the release of version 1.1.  Therefore,
 * this example can be compiled only if your C-disk supports version 1.1 or beyond.
 *****/

#include "exec/types.h"
#include "graphics/gfx.h"
#include "hardware/dmabits.h"
#include "hardware/custom.h"
#include "hardware/blit.h"
#include "graphics/gfxmacros.h"
#include "graphics/copper.h"
#include "graphics/view.h"
#include "graphics/gels.h"
#include "graphics/regions.h"
#include "graphics/clip.h"
#include "exec/exec.h"
#include "graphics/text.h"
#include "graphics/gfxbase.h"
/* ***** added for layers support ***** */
#include "graphics/layers.h"
#include "graphics/clip.h"
/* ***** added for layers support ***** */

#define DEPTH 2
#define WIDTH 320
#define HEIGHT 200
#define NOT_ENOUGH_MEMORY -1000

/* construct a simple display */
#define FLAGS LAYERSMART
/* dynamically created RastPorts from the calls to CreateUpfrontLayer */
struct RastPort *rp[3];      /* RastPort for each layer */

struct ColorMap *GetColorMap();
struct GfxBase *GfxBase;

SHORT boxoffsets[] = { 802, 2010, 3218 };
                /* black, red, green, blue */
USHORT colortable[] = { 0x000, 0xf00, 0x0f0, 0x00f };
long LayersBase;
extern struct Layer *CreateUpfrontLayer();
extern struct Layer_Info *NewLayerInfo();

```

```

main()
{
    struct View *oldview; /* save pointer to old View so can go back to sys */
    struct View v;
    struct ViewPort vp;
    struct ColorMap *cm; /* pointer to ColorMap structure, dynamic alloc */
    struct RasInfo ri;
    struct BitMap b;
    short i,j,k,n;
    struct Layer_Info *li;
    struct Layer *layer[3];

    GfxBase = (struct GfxBase *)OpenLibrary("graphics.library",0);
    if (GfxBase == NULL) exit(1);

    LayersBase = OpenLibrary("layers.library",0);
    if(LayersBase == NULL) exit(2);

    oldview = GfxBase->ActiView; /* save current View, go back later */
    /* example steals screen from Intuition */

    li = NewLayerInfo(); /* get a Layer_Info structure */
    if(li == NULL) exit(100);

    /* not needed if gotten by NewLayerInfo InitLayers(li);
       FattenLayerInfo(li); */

    InitView(&v); /* initialize View */
    v.ViewPort = &vp; /* link View into ViewPort */
    InitVPort(&vp); /* init ViewPort */
    /* now specify critical characteristics */
    vp.DWidth = WIDTH;
    vp.DHeight = HEIGHT;
    vp.RasInfo = &ri;
        /* init BitMap (for RasInfo and RastPort) */
    InitBitMap(&b,DEPTH,WIDTH,HEIGHT);
    ri.BitMap = &b; /* (init RasInfo) */
    ri.RxOffset = 0; /* align upper left corners of display
        * with upper left corner of drawing area */
    ri.RyOffset = 0;
    ri.Next = NULL;
        /* (init color table) */
    vp.ColorMap = GetColorMap(4); /* four entries, since only two planes deep */
    colorpalette = (UBYTE *)cm->ColorTable;
        /* copy my colors into this data structure */

```

```

LoadRGB4(vp,colortable,4);

/* allocate space for BitMap */
for(i=0; i<DEPTH; i++)
{
    b.Planes[i] = (PLANEPTR)AllocRaster(WIDTH,HEIGHT);
    if(b.Planes[i] == NULL) exit(NOT_ENOUGH_MEMORY);
    BltClear(b.Planes[i],RASSIZE(width,height),0);
}

MakeVPort( &v, &vp ); /* construct Copper instr (prelim) list */
MrgCop( &v ); /* merge prelim lists together into a real
               * Copper list in the View structure. */

LoadView(&v);

/* now fill some boxes so that user can see something */

/* Layer_Info, common BitMap, x,y,x2,y2,
   * flags = 0 (simple refresh), null pointer to superbitmap */
layer[0] = CreateUpfrontLayer(li,&b,5,5,85,65,FLAGS,NULL);
if(layer[0] == NULL) goto cleanup1;

layer[1] = CreateUpfrontLayer(li,&b,20,20,100,80,FLAGS,NULL);
if(layer[1] == NULL) goto cleanup2;

layer[2] = CreateUpfrontLayer(li,&b,45,45,125,105,FLAGS,NULL);
if(layer[2] == NULL) goto cleanup3;

for(i=0; i<3; i++) /* layers are created, now draw to them */
{
    rp[i] = layer[i]->rp;
    SetAPen(rp[i],i+1);
    SetDrMd(rp[i],JAM1);
    RectFill(rp[i],0,0,79,59);
}
SetAPen(rp[0],0);
Move(rp[0],5,30);
Text(rp[0],"Layer 0",7);

SetAPen(rp[1],0);
Move(rp[1],5,30);
Text(rp[1],"Layer 1",7);

SetAPen(rp[2],0);

```

```

Move(rp[2],5,30);
Text(rp[2], "Layer 2",7);

Delay(100);      /* two seconds before first change */
BehindLayer(li,layer[2]);

Delay(100);      /* another change two seconds later */

UpfrontLayer(li,layer[0]);

for(i=0; i<30; i++)
{
    MoveLayer(li,layer[1],1,3);
    Delay(10);      /* wait .2 seconds (uses DOS function) */
}

cleanup3:
    LoadView(oldview);      /* put back the old View */
    DeleteLayer(li,layer[2]);
cleanup2:
    DeleteLayer(li,layer[1]);
cleanup1:
    DeleteLayer(li,layer[0]);

    DisposeLayerInfo(li);

    /* return user and system-allocated memory to sys manager */
for(i=0; i<DEPTH; i++) /* free the drawing area */
    FreeRaster(b.Planes[i],WIDTH,HEIGHT);
FreeColorMap(cm);      /* free the color map */
/* free dynamically created structures */
FreeVPortCopLists(&vp);
FreeCprList(v.LOFCprList);
return(0);

CloseLibrary(GfxBase);

} /* end of main() */

```


Clipping Rectangle List

When you perform the various graphics drawing routines, you will notice that the routines draw into Intuition windows, even though the windows might be partially or totally obscured on the screen. This is because the layer library functions split the drawing area to provide lists of drawing areas that the graphics drawing can use for its operations.

In particular, the layer library functions split the windows into rectangles. You need only concern yourself with a single overall **RastPort** that contains the description of the complete area that you are managing. When either you or Intuition use the layer library, the graphics routines will be able to tell how the drawing area is split and where rendering can occur.

The set of rectangles comprising the layer is known as a clipping rectangle list (**ClipRect** structure). A clipping rectangle is a rectangular area into which the graphics routines will draw. All drawing that would fall outside of that rectangular area is clipped (not rendered).

DAMAGE LIST

For a smart-refresh window, the system automatically generates off-screen buffer spaces, essentially linked into the clipping rectangle list. Thus, parts of the display that are on the screen are rendered into the on-screen drawing area, and parts of the display that are obscured are drawn into a back-up area. When segments are exposed, the back-up area information is brought to view automatically during the routines **UpfrontLayer()** and **BehindLayer()**, as well as during **MoveLayer()**.

For a simple-refresh window however, any section of a drawing area that is not covered in the clipping rectangle list is not drawn into by the graphics routines. When obscured areas are exposed, they will not contain any graphics rendering at all. As the system creates and moves layers in front of such simple-refresh windows, the layers library keeps track of the rectangular segments that have not been drawn and are therefore not part of any automatically saved back-up areas. This list of non-drawn areas is called a **DamageList**.

REPAIRING THE DAMAGE

When you receive a REFRESH event from Intuition for a simple refresh window, you are being told that Intuition, through the layers library, has done something to change the portions of your window that are exposed to view. In other words, there is likely to be a blank space where there is supposed to be some graphics.

To update only those areas that need updating, you call **BeginUpdate()**. **BeginUpdate()** saves the pointer to the current clipping rectangles. It also installs in the layer structure a pointer to the set of **ClipRects** generated from the **DamageList**. In other words, the graphics rendering routines see only those rectangular spaces that need to be updated and refuse to draw into any other spaces within this layer. If, for example, there are only one or two tiny rectangles that need to be fixed, the graphics routines can ignore all but these spaces and repair them very quickly and efficiently. To repair the layer, you ask the graphics routines to redraw the whole layer, but the routines use the new clipping rectangle list (that is, the damage list) to speed the process.

To complete the update process call **EndUpdate()**, to restore the original **ClipRect** list.

Regions

Regions are rectangles that, when combined, can become part of a **DamageList**. The library *graphics.library* contains several support routines for regions. Among these are routines for the following operations:

Operation	Routine
Creating and deleting regions	NewRegion() , DisposeRegion()
Changing a region	AndRectRegion() , OrRectRegion , XorRectRegion()
Clearing a region	ClearRegion()

Basically, the region commands let you construct a custom **DamageList**, which you can use with your graphics rendering routines. With this list, you can selectively update a custom-sized, custom-shaped part of your display area without disturbing any of the other layers that might be present.

CREATING AND DELETING REGIONS

NewRegion() allocates and initializes a new data structure that may be thought of as a blank painter's easel.

If this new region is to be used as the basis for a **DamageList**, and you asked the graphics routines to draw something through this **DamageList**, nothing would be drawn as there is nothing in the region. The region that you produce can be thought of as patches of canvas. A new region has no canvas.

Because a region is dynamically created by using **NewRegion()**, the procedure **DisposeRegion()** is provided to return the memory to the system when you have finished with it. Note that not only the region structure is deallocated; so are any rectangles that have been linked into it.

CHANGING A REGION

OrRectRegion() modifies a region structure by *or*'ing a clipping rectangle into the region. This has an effect similar to adding a rectangle of canvas to the easel. If you now exercise the drawing routines, the rendering will occur in the areas where the region has been *or*'ed (canvas rectangle has been added) and will be inhibited elsewhere.

AndRectRegion() modifies a region structure by *and*'ing a clipping rectangle into the region. This has an effect similar to using the rectangle as an outline for a position on the easel. Any area of canvas that falls outside this outline is clipped and discarded.

XorRectRegion() applies the rectangle to the region in an exclusive-or mode. That is, wherever there is no canvas, canvas is applied to the easel. Wherever there is canvas present within the rectangle, a hole is created. Thus it is a combination of **OrRectRegion()** and **AndRectRegion()** in a single application.

CLEARING A REGION

While you are performing various types of selective drawing area updates, you may wish to do some of your graphics rendering with one form of region, and some with a different form of region. You can perform **ClearRegion()** to go from one form back to a fresh, empty region. Then you can begin again to compose yet another modified region for the next drawing function.

USING REGIONS

The region routines typically are used in a sequence like the following:

```

struct Region *r;
struct Rectangle *rect1, *rect2, rect3;

r = NewRegion();
OrRectRegion(rect1, r);      /* add a rectangle */
AndRectRegion(rect3, r);     /* patch a rectangle */
XorRectRegion(rect2, r);     /* weird patch */
...
/* in this section of code:
* 1. Save current pointer to DamageList for the layer you wish to affect.
* 2. Equate the region address (r) to the DamageList pointer in the
*    layer structure.
* 3. Perform whatever drawing functions you wish into this layer.
* 4. Restore the original DamageList pointer.
*/
...
DisposeRegion(r);

```

The drawing will only occur in those areas of the drawing area that you have specified should be updated. Graphics rendering is often made faster this way, because not all of the area need be updated.

A typical sequence using `ClearRegion()` might be:

```

struct Region *r;
struct Rectangle *rect1, *rect2, rect3;
struct Layer_Info *li;

r = NewRegion();
OrRectRegion(rect1, r);
OrRectRegion(rect2, r);
...
    (swap in as a damage list)
BeginUpdate(li);
    (draw, draw, draw something)
EndUpdate(li);
    (restore original damage list)
...
ClearRegion(r);
AndRectRegion(rect3, r);
...
    (swap, draw, restore)
...
DisposeRegion(r);

```

SAMPLE APPLICATION FOR REGIONS

For example, assume that you are producing a display that requires a view through a fence. You can create this “slats” effect by using regions, as follows:

1. Create a new region.
2. Create several rectangles representing the open areas of the slats in the fence.
3. *Or* these into the region.
4. Save the **DamageList** pointer in the affected layer so it can be restored later.
5. Copy the region address into **DamageList** pointer.
6. Draw the scene into the entire layer using the graphics.
7. Restore the original **DamageList** pointer.
8. Dispose of the region.

Here is a sample application. It is based on the sample layers library program shown above. For brevity, the comments have been stripped out except where new material, pertinent to regions, has been inserted.

```
/* SIMPLE REGIONS EXAMPLE.... DRAW BEHIND A FENCE */
/* Certain layers.library routines are used herein that are not
 * available until Amiga C compiler version 1.1 and beyond. */

#include <exec/types.h>
#include <graphics/gfx.h>
#include <hardware/dmabits.h>
#include <hardware/custom.h>
#include <graphics/gfxmacros.h>
#include <graphics/regions.h>
#include <graphics/clip.h>
#include <graphics/text.h>
#include <hardware/blit.h>
#include <graphics/gfxbase.h>
#include <graphics/copper.h>
#include <graphics/gels.h>
#include <graphics/rastport.h>
#include <graphics/view.h>
#include <exec/exec.h>
```

```

#include <graphics/layers.h>

#define FLAGS LAYERSIMPLE
extern struct Layer *CreateUpfrontLayer();

struct GfxBase *GfxBase;

long LayersBase;

#define DEPTH 2
#define WIDTH 320
#define HEIGHT 200
#define NOT_ENOUGH_MEMORY -1000

struct ColorMap *GetColorMap();

USHORT colortable[] = { 0x000, 0xf00, 0x0f0, 0x00f };
    /* black, red, green, blue */

extern struct Layer_Info *NewLayerInfo();

main()
{
    struct View *oldview;
    struct View v;
    struct ViewPort vp;
    struct ColorMap *cm;
    struct RasInfo ri;
    struct BitMap b;
    struct RastPort *rp;    /* one RastPort for one layer */
    short i,j,k,n;
    UBYTE *displaymem;
    UWORD *colorpalette;

    struct Layer_Info *li;
    struct Layer *layer;    /* one layer pointer */

    extern struct Region *NewRegion();
    struct Region *rgn;    /* one region pointer */
    struct Rectangle rect[14]; /* some rectangle structures */
    struct Region *oldDamageList;
    SHORT x,y;

    GfxBase = (struct GfxBase *)OpenLibrary("graphics.library",0);
    if (GfxBase == NULL) exit(1);

```

```

LayersBase = OpenLibrary("layers.library",0);
if(LayersBase == NULL) exit(2);

oldview = GfxBase->ActiView;

li = NewLayerInfo();    /* v1.1 code only */
InitView(&v);
v.ViewPort = &vp;
InitVPort(&vp);
vp.DWidth = WIDTH;
vp.DHeight = HEIGHT;
vp.RasInfo = &ri;
InitBitMap(&b,DEPTH,WIDTH,HEIGHT);
ri.BitMap = &b;
ri.RxOffset = 0;
ri.RyOffset = 0;
ri.Next = NULL;
cm = GetColorMap(4);
colorpalette = (UWORD *)cm->ColorTable;
for(i=0; i<4; i++)
{
    *colorpalette++ = colortable[i];
}
vp.ColorMap = cm;
for(i=0; i<DEPTH; i++)
{
    b.Planes[i] = (PLANEPTR)AllocRaster(WIDTH,HEIGHT);
    if(b.Planes[i] == NULL) exit(NOT_ENOUGH_MEMORY);
    BltClear(b.Planes[i],RASSIZE(WIDTH,HEIGHT),0);
}

MakeVPort( &v, &vp );
MrgCop( &v );

LoadView(&v);
layer = CreateUpfrontLayer(li,&b,0,0,200,140,FLAGS,NULL);
if(layer==NULL) exit(3);

rp = layer->rp;

SetAPen(rp,3);
RectFill(rp,0,0,199,139);    /* show the layer itself */

j=10;                        /* initialize the rectangles */

```

```

for(i=0; i<10; i++)
{
    rect[i].MinX = j;
    rect[i].MaxX = j + 8;
    rect[i].MinY = 20;
    rect[i].MaxY = 120;
    j += 16;
}

rgn = NewRegion();          /* get a new region to use */
if(rgn == NULL) exit(4);

for(i=0; i<14; i++)
    OrRectRegion(rgn,&rect[i]);

oldDamageList = layer->DamageList;
layer->DamageList = rgn;

BeginUpdate(layer);

/* here insert the drawing routines to draw something behind the slats */
x = 4; y = 10;
SetAPen(rp,0);
SetDrMd(rp,JAM1);
RectFill(rp,0,0,199,139);
SetAPen(rp,1);
SetBPen(rp,0);
SetDrMd(rp,JAM2);
for(i=0; i<14; i++)
{
    Move(rp, x, y);
    Text(rp,"Behind A Fence",14);
    x += 4; y += 9;
}
EndUpdate(layer);
layer->DamageList = oldDamageList;
DisposeRegion(rgn);

Delay(300);

DeleteLayer(li, layer);
DisposeLayerInfo(li);

LoadView(oldview);

```



```
    /* return user and system-allocated memory to sys manager */  
for(i=0; i<DEPTH; i++)/* free the drawing area */  
    FreeRaster(b.Planes[i],WIDTH,HEIGHT);  
FreeColorMap(cm);          /* free the color map */  
                            /* free dynamically created structures */  
FreeVPortCopLists(&vp);  
FreeCprList(v.LOFCprList);  
return(0);  
  
CloseLibrary(GfxBase);  
  
} /* end of main() */
```

Chapter 3

Animation

Introduction

The graphics animation routines let you define images by specifying various characteristics of graphic objects, such as the following:

- o Height
- o Width
- o Colors
- o Shape
- o Position in the drawing area
- o How to draw the object
- o How to move the object
- o How the object interacts with other elements

The objects you define are called GELS (for “graphic elements”). You can draw GELS into or onto a background display of some type. The graphics animation routines operate on a list of GELS to produce a list of instructions that cause the system to draw the GELS in the manner you have specified.

PREPARING TO USE GRAPHICS ANIMATION

Because the animation routines have been designed to interact with a background display, you must first make sure that such a display is already defined.

To define a display with which the GELS can interact, you define **View**, **ViewPort**, and **RastPort** structures. For details on the construction of these structures, see chapter 1, “Graphics Primitives,” and chapter 2, “Layers.”

The graphics animation routines described in this chapter create additional material that is linked into the **View** structure. This material consists of additional instructions for color changes and dynamic reassignment of the hardware resources that create the display animation effects you specify.

TYPES OF ANIMATION

Using the Amiga system tools, you can perform two different kinds of image animation: sprite animation and playfield animation.

Sprite Animation

Sprites are hardware objects that you create and move independently of the playfield display. Sprites are always 16 low-resolution pixels wide and are as high as you specify. To move sprites, you must define where they are on the screen. The built-in priority circuitry determines how the sprite appears on the screen relative to the playfield elements or to other sprites.

You can manipulate sprites directly through a simple sprite set of routines or by using the graphics kernel **VSprite** routines.

Playfield Animation

Sprites are normally moved against a background. This background area is called the playfield. You may treat the playfield area as a single background or separate it into two separately controllable sections, using dual-playfield mode. See chapter 1, "Graphics Primitives," for details on how to create and control playfields.

In playfield animation, sections of the playfield are modified. You draw, erase, and redraw objects into the playfield, creating an animation effect. To move the data quickly and efficiently, the system uses one of the specialized built-in hardware devices, the *blitter*. The system uses the blitter to move the playfield objects, while it saves and restores the background. The objects controlled by the blitter are called **Bobs**, for "blitter objects."

Playfield animation is somewhat more complicated than **VSprite** animation from the point of view of system design, but not much more complicated for you as the user of the animation routines. The hardware displays the **VSprites** over the playfield automatically, and the priority overlay circuitry assures that they will be displayed in the correct order. If you are animating multiple **Bobs**, you control their *video priority* by defining the sequence in which the system draws them. The last one drawn has the highest video priority in the sense that it appears to be in front of all other **Bobs**.

A **Bob** is physically a part of the playfield. When the system displays a **Bob**, it must first save a copy of the playfield area into which the **Bob** will be drawn. Then the system can restore the playfield to its original condition when moving the **Bob** to a new location. Once the playfield areas have been saved, the system can draw the **Bob**. To move the **Bob**, the system must first restore the playfield area (thus erasing the object) before it saves the playfield at the new location and draws the **Bob** there.

Bobs offer more flexibility and many more features than **VSprites**. **Bob** animation is less restrictive but slower than **VSprite** animation. **VSprites** are superior to **Bobs** in speed of display, because **VSprites** are mostly hardware-driven and **Bobs** are part hardware and part software. **Bobs**, on the other hand, are superior to **VSprites** in that they offer almost all of the benefits of **VSprites** but suffer none of the limitations, such as size or number of colors.

Both are very powerful and useful. The requirements of your particular application determine the type of GEL to use.

THE GELS SYSTEM

The acronym GEL describes all of the graphic elements, or “objects,” supplied by the Amiga ROM kernel. Both **VSprites** and **Bobs** are GELS, as are the more advanced animation elements known as **AnimComps** and **AnimObs**.

Initializing the GEL System

To initialize the graphics element animation system, you provide the system with the addresses of two data structures. The system uses these data structures to keep track of the GELS that you will later define. To perform this initialization, you call the system routine **InitGels()**, which takes the form:

```
InitGels( head, tail, Ginfo );
```

where

head

is a pointer to the **VSprite** structure to be used as the GEL list head

tail

is a pointer to the **VSprite** structure to be used as the GEL list tail

Ginfo

is a pointer to the **GelsInfo** structure to be initialized

The graphics animation system uses two “dummy” **VSprites** as place holders in the list of GELS that you will construct. The dummy **VSprites** are used as the head and tail elements in the system list of GELS. You add graphics elements to or delete them from this list.

The call to **InitGels()** forms a linked list of GELS that is empty except for these two dummy elements. When the system initializes the list with the dummy **VSprite**, it automatically gives the **VSprite** at the head the maximum possible negative y and x positions and the **VSprite** at the tail the maximum possible positive y and x positions. This assures that the two dummy elements are always the outermost elements of the list.

The y,x values are coordinates that relate to the physical position of the GEL within the drawing area. The system uses the y,x values as the basis for the placement (and later sorting) of the GELS in the list.

When you add a GEL to the list of graphics elements, the system links that GEL into the list shown above. Then the system adds any new element to the list immediately ahead of the first GEL whose y,x value is greater than or equal to that of the new GEL being added.

Types of GELS

Figure 3-1 shows how you can view the components of GELS as inter-related layers of graphics elements.

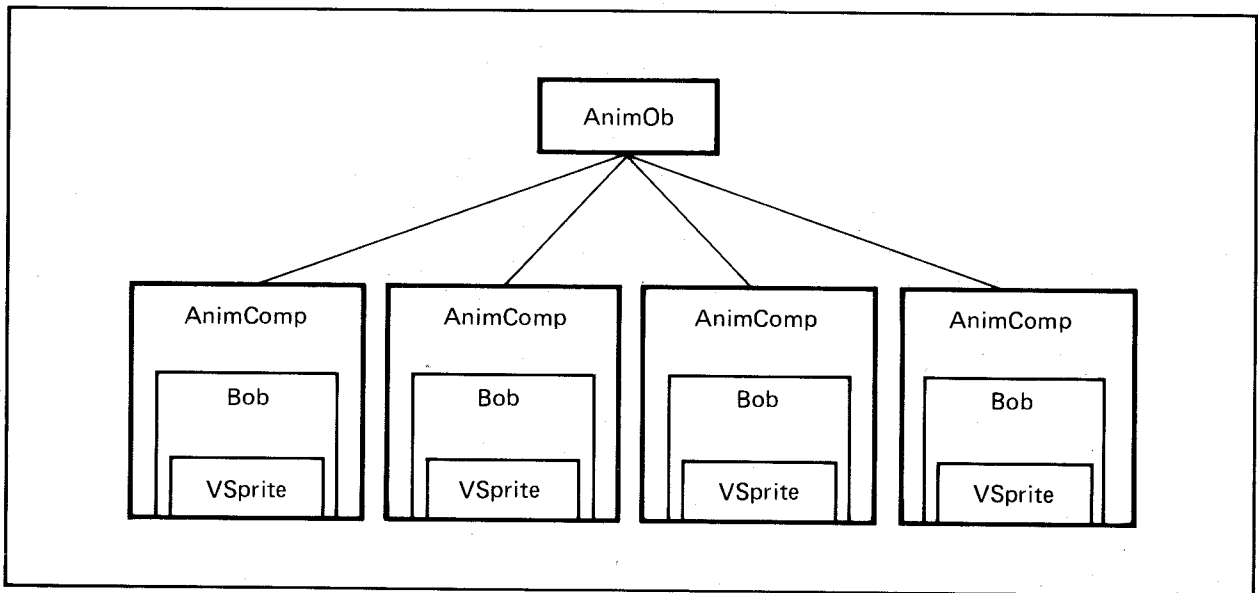


Figure 3-1: Shells of Gels

The types of GELS are listed below:

- o Simple (hardware) sprites
- o **VSprites**
- o **Bobs**
- o **AnimComps**
- o **AnimObs**

VSprites and **Bobs** are the primary software-controlled animation objects. They are part of an integrated animation system. The simple sprites, on the other hand, are separate from the animation system. It is up to you to decide which type of sprite to use. The next sections describe all of these animation components.

Simple (Hardware) Sprites

The simple sprite is a special graphics element, related to the graphics animation system only in that it vies with the **VSprites** for the use of the same underlying hardware elements, the real hardware sprites.

The Amiga hardware has the ability to handle up to eight sprite objects. Each sprite is produced by one of the eight hardware sprite DMA channels. Each sprite is 16-bits wide and arbitrarily tall. The Amiga software provides a choice about how you can use these hardware elements. You can either allocate one or more hardware sprites for your exclusive use, or you can allow all sprites to be managed by the system software and assigned as virtual sprites by the system. Using virtual sprites, it can appear as though you have an unlimited set of sprites with which to work. If you need only a few sprites, however, you may wish to use the less complex routines shown in the section called "Using Simple Sprites."

VSprites

The virtual sprite is the most elemental component. It contains a little more information than is needed to define a hardware sprite. The system temporarily assigns each **VSprite** to a hardware sprite, as needed. The information in the **VSprite** structure allows the system to maintain the more general GEL functions, such as collision detection and double-buffering. After a sprite DMA channel has displayed the last line of a sprite, the system can reuse the channel to display a different image lower on the screen. The system software takes advantage of this reusability to dynamically assign hardware sprites to carry **VSprite** images.

The **VSprite** is a data structure closely related to hardware sprites. The **VSprite** structure contains the following information:

- o Size
- o Image display data
- o Screen coordinates
- o Collision descriptors
- o A pointer to color information

Bobs

The **Bob** is the next outermost level of the GEL system. It is like an expanded hardware sprite done in software. It uses the same information defined in a **VSprite**, but adds other data that further defines this type of object. **Bobs** and **VSprites** differ in that the system draws **Bobs** into the playfield using the blitter, while it assigns **VSprites** to hardware sprites.

A **Bob** structure contains the following information:

- o A pointer to a **VSprite**
- o Priority descriptors
- o Variables and pointers that define how and where to save the background

AnimComps

The **AnimComp** (for “animation component”) is a data structure that extends the definition of a **Bob**. It allows the system to include the **Bob** as part of a total animation object. An **AnimComp** expands on the **Bob** data. **AnimComps** include the following:

- o A pointer to this **AnimComp**'s **Bob**
- o Links that define the sequence of animation drawings
- o Information that describes the screen coordinates of the **AnimComp** with respect to the position of the **AnimOb**, described below
- o Timing information for sequencing this **AnimComp** as part of the list of animation drawings
- o A pointer to a user routine to execute in conjunction with this **AnimComp**

AnimObs

The **AnimOb** (for “animation object”) is the primary animation object. It is a pseudo-object whose primary purpose is to link one or more **AnimComps** into a single overall object. As the **AnimOb** moves, so move its **AnimComps**. When the **Bobs** move with their **AnimComps**, the system sets the screen coordinates in the **VSprite** accordingly. **AnimObs** include the following:

- A pointer to this **AnimOb**'s first **AnimComp**
- Links to previous or succeeding **AnimObs**
- Information that describes the position of this **AnimOb** on the screen, as well as its velocity and acceleration
- Information for double-buffering this **AnimOb**, if desired
- A pointer to a user routine to execute in conjunction with this **AnimOb**

Using Simple (Hardware) Sprites

To use simple sprites, define their data structures and use the following routines:

- **ON_SPRITE** — a system macro to turn on sprite DMA
- **OFF_SPRITE** — a system macro to turn off sprite DMA
- **GetSprite()** — attempts to allocate a sprite from the virtual sprite machine for your exclusive use
- **ChangeSprite()** — modifies the sprite's appearance
- **MoveSprite()** — changes the sprite's position
- **FreeSprite()** — returns the sprite to the virtual sprite machine

These routines are described in detail in the following sections.

To use these simple sprite routines or the **VSprite** routines, you must include the **SPRITE** flag in the data structure for **OpenScreen()**. If you are not using Intuition, this flag must be specified in the **View** and **ViewPort** data structures before **MakeView()** is called.

CONTROLLING SPRITE DMA

You can use the graphics macros **ON_SPRITE** and **OFF_SPRITE** to control sprite DMA. **OFF_SPRITE** prevents the system from displaying any sprites, whether hardware or **VSprite**. **ON_SPRITE** restores the sprite data access and display. Note that the Intuition cursor is a sprite. Thus, if you use **OFF_SPRITE**, you make Intuition's cursor invisible as well.

ACCESSING A HARDWARE SPRITE

You use `GetSprite()` to gain access to a new hardware sprite. You use a call such as

```
status = GetSprite( sprite, number )
```

`GetSprite()` allocates a hardware sprite for your exclusive use. The virtual sprite allocator can no longer assign this sprite. Note that if you steal one sprite, you are effectively stealing two. The sprite pairs 0/1, 2/3, 4/5, and 6/7 share the same color registers. If you are stealing a hardware sprite, you steal its color registers as well. So you might as well ask for the other sprite in the pair. Table 3-1 shows the color registers assigned to each sprite pair.

Table 3-1: Sprite Color Registers

Color Registers	Sprite
16-19	0 or 1
20-23	2 or 3
24-27	4 or 5
28-31	6 or 7

You are not granted *exclusive* use of the color registers. If the **ViewPort** is 5 bit-planes deep, all 32 of the system color registers will still be used by the playfield display hardware.

Note, however, that registers 16, 20, 24, and 28 always generate the “transparent” color when selected by a sprite, regardless of which color is actually in them. Their true color will be used only if they are selected by a playfield. For further information, see the *Amiga Hardware Reference Manual*.

Also note that sprites and sprite colors are bound to the **ViewPort** in that you can reload the colors between **ViewPorts**. In other words, if a user in a **ViewPort** located in the top part of the screen allocates sprite 0 and a user in the a **ViewPort** at the bottom of the screen allocates sprite 1, these two sprites will not necessarily have the same color set, as the two **ViewPorts** can have totally independent sets of colors.

The inputs to the `GetSprite()` routine are:

- sprite** A pointer containing the address of a data structure called **SimpleSprite**
- number** The number (0-7) of the hardware sprite you wish to reserve. If **number** is -1, the system gets any sprite.

A value of 0-7 is returned in "status" if your request was granted, specifying which sprite you have allocated. A value of -1 means that this sprite is already allocated.

The structure for a simple sprite is shown below:

```
struct SimpleSprite {
    /* pointer to definition data of the hardware sprite to be displayed */
    UWORD *posctldata;
    UWORD height; /* height of this simple sprite in rows */
    UWORD x,y; /* current position */
    /* number (0-7) of hardware sprite associated with this simple sprite */
    UWORD num;
};
```

This data structure is found in the *graphics/sprite.h* file in the appendixes to this manual.

CHANGING THE APPEARANCE OF A SIMPLE SPRITE

The **ChangeSprite()** routine changes the appearance of a reserved sprite. It is called by the following sequence:

ChangeSprite(vp, s, newdata)

ChangeSprite() substitutes a new data content for that currently used to display a reserved hardware sprite.

The inputs to this routine are:

- vp** A pointer to the **ViewPort** for this sprite or 0 if this sprite is relative only to the current **View**
- s** A pointer to a **SimpleSprite** structure
- newdata** A pointer to a data structure containing the new data to be used

The structure for the new data is shown below:

```

struct userspritedata
{
    /* position and control information for this sprite */
    UWORD posctl[2];
    /* two words per line of sprite height, first of the two
    * words contains msbit for color selection, second word
    * contains lsbit (colors 0,1,2,3 from allowable color
    * register selection set). Color '0' for any sprite
    * pixel makes it transparent.
    */
    UWORD sprdata[2][height];      /* actual sprite image */

    /* initialize to 0, 0 for unattached simple spites */
    UWORD reserved[2];
};

```

MOVING A SIMPLE SPRITE

MoveSprite() repositions a reserved hardware sprite. It is called as follows:

```
MoveSprite( vp, sprite, x, y )
```

After you call this routine, the reserved sprite is moved to a new position relative to the upper left corner of the **ViewPort**.

The inputs to **MoveSprite()** are as follows:

- vp** A pointer to the **ViewPort** with which this sprite interacts or 0 if this sprite's position is relative only to the current **View**
- sprite** A pointer to a **SimpleSprite** structure
- x, y** Pixel position to which a sprite is to be moved. If the sprite is being moved over a high-resolution display, the system can move the sprite only in two-pixel increments. In low-resolution mode, single-pixel increments in the x direction are acceptable. For an interlaced mode display, the y direction motions are in two line increments. The same image of the sprite is placed into both even and odd fields of the interlaced display.

The upper left corner of the **ViewPort** area has coordinates (0,0). The motion of the sprite is relative to this position.

The following example demonstrates how you move a simple sprite.

```
/* This program creates and displays a 320-by-200 by 2-bit-plane
 * single-playfield display and adds one simple sprite to it.
 */

#include "exec/types.h"
#include "graphics/gfx.h"
#include "hardware/dmabits.h"
#include "hardware/custom.h"
#include "hardware/blit.h"
#include "graphics/gfxmacros.h"
#include "graphics/copper.h"
#include "graphics/view.h"
#include "graphics/gels.h"
#include "graphics/regions.h"
#include "graphics/clip.h"
#include "exec/exec.h"
#include "graphics/text.h"
#include "graphics/gfxbase.h"
#include "graphics/sprite.h"

#define DEPTH 2
#define WIDTH 320
#define HEIGHT 200
#define NOT_ENOUGH_MEMORY -1000

/* construct a simple display */

struct View view;
struct ViewPort viewport;

/* pointer to ColorMap structure, dynamically allocated */
struct ColorMap *cm;

struct RasInfo rasinfo;
struct BitMap bitmap;

SHORT xmove, ymove;

extern struct ColorMap *GetColorMap();
struct GfxBase *GfxBase;

/* save pointer to old View so can restore */
struct View *oldview;
```

```

USHORT colortable[] = {
    /* black, red, green, blue */
    0x000, 0xf00, 0x0f0, 0x00f,
    0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0, /* sprites from here up */
    0,0,0,0,0,0,0,0
};

/* where to draw boxes */
SHORT boxoffsets[] = {
    802, 2010, 3218
};

UWORD *colorpalette;

struct SimpleSprite sprite;

/* Last entry is "position control" for the next reuse of the hardware sprite.
 * Simple sprite machine supports only one use of a hardware sprite per video
 * frame. Any combination of binary bits from word 1 and word 2 per line
 * establishes the color for a pixel on that line. Any nonzero pixels in lines
 * 1-3 are color "1" of the sprite, lines 4-6 are color "2", lines 7-9 are color "3".
 */
UWORD sprite_data[] = {
    0,0, /* position control */
    0x0fc3, 0x0000, /* image data line 1 */
    0x3ff3, 0x0000, /* image data line 2 */
    0x30c3, 0x0000, /* image data line 3 */
    0x0000, 0x3c03, /* image data line 4 */
    0x0000, 0x3fc3, /* image data line 5 */
    0x0000, 0x03c3, /* image data line 6 */
    0xc033, 0xc033, /* image data line 7 */
    0xffc0, 0xffc0, /* image data line 8 */
    0x3f03, 0x3f03, /* image data line 9 */

    /* NOTE this last line specifies unattached, simple sprites */
    0, 0 /* next sprite field */
};

/*****
 * FOLLOWING IS FOR INFORMATION ONLY... the simple-sprite machine directly
 * sets these bits; the user has no need to change any of them. Use the
 * functions ChangeSprite() and MoveSprite() to have an effect on the sprite.
 *

```

```

* position control:
*
* first UWORD:
*   bits 15-8, start vertical value, lowest 8 bits of this value
*   contained here.
*   bits 7-0, start horizontal value, highest 8 bits of this value
*   contained here.
*
* second UWORD:
*   bits 15-8, end (stopping) vertical value, lowest 8 bits of this
*   value contained here.
*   bit 7 = Attach-bit (used for attaching sprites to get additional
*   colors (15 instead of 3, supported by the hardware but
*   NOT supported by the simple sprite machine).
*   bits 6-4 (unused)
*   bit 2 start vertical value; bit 8 of that value.
*   bit 2 end vertical value; bit 8 of that value.
*   bit 2 start horizontal value; bit 0 of that value.
*
*****/

```

```

main()
{
    LONG i;
    SHORT j,k,n;
    SHORT spgot;
    UBYTE *displaymem;

    GfxBase = (struct GfxBase *)OpenLibrary( "graphics.library" , 0 );
    if( GfxBase == NULL ) exit(100);

    /* save current view to restore later */
    oldview = GfxBase->ActiView;

    /* example steals screen from Intuition if started from WBench */

    InitView( &view );           /* initialize View */
    InitVPort( &viewport );      /* init ViewPort */
    view.ViewPort = &viewport;  /* link View into ViewPort */

    /* init bit map (for RasInfo and RastPort) */
    InitBitMap( &bitmap, DEPTH, WIDTH, HEIGHT );

    /* init RasInfo */
    rasinfo.BitMap = &bitmap;

```

```

rasinfo.RxOffset = 0;
rasinfo.RyOffset = 0;
rasinfo.Next = NULL;

/* now specify critical characteristics */
viewport.DWidth = WIDTH;
viewport.DHeight = HEIGHT;
viewport.RasInfo = &rasinfo;

/* initialize the color map. It has 32 entries. Sprites take up
 *the top 16 and we want to specify some sprite colors */
cm = GetColorMap( 32 );

/* no memory for color map */
if(cm == NULL) {
    FreeMemory();
    exit( 100 );
}

colorpalette = (UWORD *)cm->ColorTable;
for(i=0; i<32; i++) {
    *colorpalette++ = colortable[i];
}

/* copy my colors into this ViewPort structure */
viewport.ColorMap = cm;

/* addition for simple sprite: */
vp.Modes = SPRITES;

/* allocate space for bitmap */
for(i=0; i<DEPTH; i++) {
    bitmap.Planes[i] = (PLANEPTR) AllocRaster( WIDTH, HEIGHT );
    if( bitmap.Planes[i] == NULL ) exit( NOT_ENOUGH_MEMORY );

    /* clear the display area */
    BltClear( bitmap.Planes[i], RASSIZE(WIDTH,HEIGHT), 1 );
}

/* construct Copper instr (prelim) list */
MakeVPort( &view, &viewport );

/* merge prelim lists into a real Copper list in the view structure. */
MrgCop( &view );
LoadView( &view );

```



```

/* now fill some boxes so that user can see something */
/* always draw into both planes to assure true colors */
for(n=1; n<4; n++) /* three boxes */
{
    for(k=0; k<2; k++)
    {
        /* boxes will be in red, green and blue */
        displaymem = bitmap.Planes[k] + boxoffsets[n-1];
        DrawFilledBox( n, k, displaymem );
    }
}

/*****
* now we are ready to play with the sprites!
*****/

/* Get the next available sprite. We should do an error
* check, if returns -1, then no sprites are available
*/
spgot = GetSprite( &sprite, -1 );

sprite.x = 0; /* initialize position and size info */
sprite.y = 0; /* matches that shown in sprite_data */
sprite.height = 9; /* so that system knows layout of data later */

/* now put some colors into this sprite's color registers
* to custom-control the colors this particular sprite will display.
* NOTE: sprite pairs share color registers; i.e., sprites 0 and 1,
* 2 and 3, 4 and 5, 6 and 7 as pairs share the same sets of color
* registers (see the Amiga Hardware Reference manual for details).
* The code following figures out which sprite the system gave us,
** and sets that sprite's color registers to the correct value
*/
k = ((spgot & 0x06)*2) + 16;

/* convert sprite number into the base number for its color reg set */
/* value at k treated as transparent */
SetRGB4( &viewport, k+1, 12, 3, 8 );
SetRGB4( &viewport, k+2, 13, 13, 13 );
SetRGB4( &viewport, k+3, 4, 4, 15 );

/* top of sprite is red, middle is white, bottom is blueish */
ChangeSprite(&viewport,&sprite, sprite_data);

MoveSprite(0,&sprite,30,0);

```

```

xmove = 1; ymove = 1;
for( n = 0; n < 4; n++ ) {
    i=0;
    while( i++ < 185 ) {
        MoveSprite( 0, &sprite, sprite.x + xmove, sprite.y + ymove );

        /* slow it down to one move per video frame */
        WaitTOF();
    }
    ymove = -ymove;
    xmove = -xmove;
}

/* free this sprite so others can use it also */
FreeSprite( spgot );

/* restore the system to its original state */
LoadView( oldview );
FreeMemory();
CloseLibrary( GfxBase );
} /* end of main() */

/* return user and system-allocated memory to sys manager */
FreeMemory()
{
    LONG i;

    /* free drawing area */
    for( i=0; i<DEPTH; i++ ) {
        if( bitmap.Planes[i] != NULL ) {
            FreeRaster( bitmap.Planes[i], WIDTH, HEIGHT );
        }
    }
    /* free the color map created by GetColorMap() */
    if( cm != NULL ) FreeColorMap( cm );

    /* free dynamically created structures */
    FreeVPortCopLists( &viewport );
    FreeCprList( view.LOFCprList );
    return( 0 );
}

```

```

DrawFilledBox( fillcolor, plane, displaymem )

```

```

SHORT fillcolor,plane;
UBYTE *displaymem;
{
    UBYTE value;
    LONG j;

    for(j=0; j<100; j++) {
        if((fillcolor & (1 << plane)) != 0) {
            value = 0xff;
        } else {
            value = 0;
        }
        for(i=0; i<20; i++) {
            *displaymem++ = value;
        }
        displaymem += (bitmap.BytesPerRow - 20);
    }
    return(0);
}

```

RELINQUISHING A SIMPLE SPRITE

The **FreeSprite()** routine returns an allocated sprite to the virtual sprite machine. The virtual sprite machine can now reuse this sprite to allocate virtual sprites. The syntax of this routine is

```
FreeSprite( num )
```

where **num** is the number (0-7) of the sprite you want to return.

Note: You *must* free sprites after you have allocated them using **GetSprite()**. If you do not free them and your task ends, the system will have no way of reallocating those sprites until the system is rebooted.

Using VSprites

This section tells how to define a **VSprite**. It describes how to:

- o Specify the size of the **VSprite** object

- o Select its colors
- o Form its image
- o Specify its position within the drawing area
- o Add it to the list of GELS
- o Control it after you add it to the list

The system software also provides a way to detect collisions between individual **VSprites** and other on-screen objects. Collision detection applies to both **VSprites** and to **Bobs**. It appears as a separate topic under "Topics Common to Both VSprites and Bobs."

SPECIFYING THE SIZE OF A VSPRITE

The first step in defining a **VSprite** is telling its dimensions to the system. A **VSprite** is always 16 pixels wide and may be any number of lines high. Each pixel is the same size as a pixel in low-resolution mode (320 pixels across a horizontal line) of the graphics display. To specify how many lines make up the **VSprite** image, you use the **VSprite** structure **Height** variable.

If your **VSprite** is 12 lines high and the name of your **VSprite** structure is **myVSprite**, then you can set the height value with the following statement:

```
myVSprite.Height = 12;
```

Each line of a **VSprite** requires two data words to specify the color content of each pixel. This means that the data area containing the **VSprite** image is 12 x 2, or 24, words long.

See the next section for details on how bits of these data words select the color of the **VSprite** pixels.

SPECIFYING THE COLORS OF A VSPRITE

Because **VSprites** are so closely related to the hardware sprites, the choice of colors for **VSprites** is limited in the same way. Specifically, each pixel of a **VSprite** can be any one of three different colors or it may be transparent. However, the system software provides a great deal of versatility in the choice of colors for the virtual sprites. Each virtual sprite may have its own set of three unique colors.

When the system assigns a hardware sprite to carry the **VSprite**'s image, it assigns that **VSprite**'s color set to the hardware sprite that will produce that image. To define which set of three colors to use for this **VSprite**, you initialize the **VSprite** structure pointer named **SprColors**. **SprColors** points to the first data item of three sequentially-stored 16-bit values. The system then jams these values into the selected hardware sprite's color registers when it is being used to display this **VSprite**.

Every time you direct the system to redraw the **VSprites**, the GEL system reevaluates the current on-screen position of each **VSprite** and decides which hardware sprite will carry this **VSprite**'s image for this rendering. It creates a customized Copper instruction sequence including both the repositioning of hardware sprites and the reloading of sprite color registers for various screen positions. Thus, during a move sequence, a **VSprite** may be represented by one or many different real hardware sprites, depending on its current position relative to other **VSprites**.

For example, if your set of colors is defined by the statement:

```
WORD spriteColors = { 0x00F, 0x0F0, 0xF00 };
```

and if your **VSprite** is named **myVSprite**, to set the **VSprite** colors you would use the following statement:

```
myVSprite.SprColors = &spriteColors;
```

How you specify the **VSprite** colors may affect how many **VSprites** you can show on the screen at any one time. For further information, see "How **VSprites** are Assigned."

SPECIFYING THE SHAPE OF A VSPRITE

To define the appearance of a **VSprite**, initialize the **VSprite** structure pointer called **ImageData** to point to the first word of the image data. A **VSprite** image is defined exactly as the image of a real hardware sprite. It takes two sequential 16-bit data words to define each line of a **VSprite**.

To select colors for the pixels of a **VSprite**, examine the combination of the data bits in corresponding locations in each of the two data words that define each line. The first of each pair of data words supplies the low-order bit of the color selector for that pixel; the second word of the pair supplies the high-order bit.

For example:

```

mem      0101111111111111
mem + 1  0011111111111111

```

Reading from left to right, the combinations of these two sequential memory data words form the binary values of 00, 01, 10, 11, and so on. These binary values select colors as follows.

- 00 - selects **VSprite** color of "transparent"
- 01 - selects the first of three **VSprite** colors you have defined
- 10 - selects the second **VSprite** color
- 11 - selects the third **VSprite** color

In those areas where the combination of bits yields a value of 00, the **VSprite** is transparent. Any object whose priority is lower than that of the **VSprite** will show through in transparent sections of the **VSprite**. Thus, you might form a full three-color image, with some transparent areas, from a data set like the following sample:

VSprite Data

mem	1111111111111111	Defines top line -
mem + 1	1111111111111111	contains only color 3
mem + 2	0011111111111100	Defines second line -
mem + 3	0011000000001100	contains colors 1 and 3 and some transparency
mem + 4	0000110000110000	Defines third line -
mem + 5	00001111111110000	contains colors 2 and 3 and some transparency
mem + 6	0000001001000000	Defines fourth line -
mem + 7	0000001111000000	contains colors 2 and 3 and some transparency
mem + 8	0000000110000000	Defines last line -
mem + 9	0000000110000000	contains color 3 and some transparency

The **VSprite** Height for this sample image is 5.

SprColors must point to the set of three colors that are to be used to display this **VSprite**, and **ImageData** must point to the location ("mem" in the example) that contains the first word of the **VSprite** definition.

SPECIFYING VSPRITE POSITION

To control the position of a **VSprite**, you use the *y* and *x* variables within the **VSprite** structure. You specify the position of the upper left corner of a **VSprite** relative to the upper left corner of the drawing area where you wish the **VSprite** to appear. Assign a value of 0,0 for *y,x* to make the **VSprite** appear with its upper left corner against the upper left corner of the drawing area. You can use values of *y* and *x* to move the **VSprite** entirely off the screen, if you wish.

You resolve the vertical positioning for **VSprites** in terms of the non-interlaced mode of the display. When you position a **VSprite** so that its *y* value is within the visible area of the screen, you can select any one of 200 possible positions down the screen at which its topmost edge can be placed.

You resolve the horizontal positioning for **VSprites** in terms of the low-resolution mode of the screen display. When you position a **VSprite** so that its *x* value is within the visible area of the screen, you can select any one of 320 possible positions across the screen at which its leftmost edge can be placed. Note that if you are using **VSprites** under Intuition and within a screen, they will be positioned relative to the upper left-hand corner of the screen.

USING VSPRITE FLAGS

Now that you have defined the **VSprite**'s size, colors, shape, and position, you may want to know where to add information to the data structures or where to check about the progress of the system routines. The following sections describe the functions of the **VSprite** flags, the variables that let you do some of these activities.

The **VSprite** data structure contains a variable named **Flags** that has information about its data and about the progress of the system routines. The following sections describe the uses of the **VSPRITE**, **VSOVERFLOW**, and **GELGONE** flags. You can use these flags to perform these tasks:

VSPRITE	Indicate whether the system should treat the structure as a VSprite or part of a Bob .
VSOVERFLOW	Check on the VSprites the system cannot display. (This is a read-only system variable.)
GELGONE	Find out if the system has moved a GEL outside the clipping region of the drawing area. (This is a read-only system variable.)

VSPRITE Flag

To tell the GEL routines to treat this **VSprite** structure as a **VSprite** instead of a **Bob**, set the **VSPRITE** flag to 1. This affects the interpretation of the data layout and the use of various system variables. If you set the **VSPRITE** flag bit to zero, the GEL routines treat this **VSprite** structure as though it defined a **Bob** instead of a **VSprite**.

Note: Under Intuition, **VSprites** work only in screens, not in windows. **Bobs** work in both screens and in windows. Thus, if you wish to use **VSprites** and **Bobs** together, you can only do so by writing directly to the **RastPort** of a screen.

VSOVERFLOW Flag

If you have currently defined more **VSprites** at the same horizontal line than the system can possibly assign to the real hardware sprites, then the **VSprites** that the system cannot display have their **VSOVERFLOW** flag set. This means that it is possible that one or more **VSprites** will not appear on the display for this pass of producing the GELS.

GELGONE Flag

When the **GELGONE** flag is set to 1, you know that the system has moved a GEL (**VSprite** or a **Bob**) entirely outside of the clipping region of the drawing area. You can assume that the system will fully or at least partially draw any objects within the clipping region. Because the system will not draw this object that is outside the clipping area, you may wish to use **RemVSprite()** to delete the **VSprite** from the GEL list in order to speed up processing of the rest of the list. Of course, **VSprites** that you remove from the list are no longer managed or checked by the system.

ADDING A VSPRITE

To control **VSprites**, you first describe them using the **VSprite** structure variables mentioned above. Next you tell the system (by adding the **VSprites** to the GEL list) which **VSprites** to handle. This section tells you how to add a **VSprite** to the GEL list.

To add a **VSprite** to the system GEL list, call the system routine **AddVSprite()**, and specify the address of the **VSprite** structure that controls this **VSprite** as well as the **RastPort** with which it is associated.

A typical system call for this purpose follows:

```
struct VSprite myVSprite;  
...  
...  
AddVSprite( &myVSprite, &rastport );
```

REMOVING A VSPRITE

To remove a **VSprite** from the list of controlled objects, use the system routine **RemVSprite()**. This function takes the following form:

```
RemVSprite( VS );
```

where **VS** is a pointer to the **VSprite** structure to be removed from the GEL list

GETTING THE VSPRITE LIST IN ORDER

When the system has displayed the last line of a **VSprite**, it reassigns the hardware sprite to another **VSprite** located at a lower position, farther left on the screen. The system allocates hardware sprites in the order in which it encounters the **VSprites** in the list. Therefore, you must sort the list of **VSprites** before the system can assign the use of the hardware sprites correctly.

When you first enter **VSprites** into the list using **AddVSprite()**, the system uses the y,x coordinates to place the **VSprites** into the correct position in the list. If you change the y,x coordinates after they are in the list, you must reorder the list before the system can use it to produce the display.

You use the routine **SortGList()** (for "sort the GEL list") to get them in the correct order before asking the system to display them. This sorting step is essential! You call this function as follows:

```
SortGList( RPort );
```

where **RPort** is a pointer to the **RastPort** structure containing the **GelsInfo**

Note that there may be a GEL list in more than one **RastPort**. You must sort all of them.

DISPLAYING THE VSPRITES

The next few sections explain how to display the **VSprites**. You use the following system routines:

- o **ON_DISPLAY** — to turn on the playfield display
- o **ON_SPRITE** — to turn on the **VSprites** display
- o **DrawGList()** — to draw the elements into the current **RastPort**
- o **MrgCop()** — to install the **VSprites** into the display
- o **LoadView()** — to ask the system to display the new **View**
- o **WaitTOF()** — to synchronize the routines with the display

Turning on the Display

Before you can view a display on the screen, you must enable the system direct memory access for both the hardware sprites and the playfield display. To enable the display of both playfield and **VSprites**, use the system macro calls:

```
ON_DISPLAY;  
ON_SPRITE;
```

Drawing the Graphics Elements

The system routine called **DrawGList()** looks through the list of controlled GELS. It prepares necessary instructions and memory areas to display the data according to your requirements. You call this routine as follows:

```
DrawGList( RPort, VPort );
```

where

RPort
is a pointer to the **RastPort**

VPort

is a pointer to the **View**

Because the system links **VSprites** to a **View**, the use of a **RastPort** is not significant for them. However, you can use **DrawGList()** for **Bobs** as well as **VSprites**, so it is required that you pass the pointer to the **RastPort** to the routine. **DrawGList()** actually draws **Bobs** into that **RastPort** when you execute the instructions.

Once **DrawGList()** has prepared the necessary instructions and memory areas to display the data, you will need to install the **VSprites** into the display with **MrgCop()**.

Merging VSprite Instructions

Recall that the call to **DrawGList()** did not actually draw the **VSprites**. It simply provided a new set of instructions that the system uses to assign the **VSprite** images to real hardware sprites, based on their positions. The **View** structure already has a set of instructions that specifies how to construct the display area. It includes pointers to the set of **VSprite** instructions that was made by the call to **DrawGList()**. To install the current **VSprites** into the display area, you call the routine **MrgCop()** to merge together all of the display-type instructions in the **View** structure. You call this routine as follows:

```
MrgCop( View );
```

where **View** is a pointer to the **View** structure whose Copper instructions are to be merged

DrawGList() handles **Bobs** as well as **VSprites**. Therefore, the call to **DrawGList()**, although it did not really draw the **VSprite** images yet, *does* draw the **Bobs** into the selected **RastPort**.

Loading the New View

Now that the display instructions include the definition of the **VSprites**, you can ask the system to prepare to display this newly configured **View**. You do this with the following system routine:

```
LoadView( view );
```

where **view** is a pointer to the **View** that contains the pointer to the Copper instruction list

The Copper instruction lists are double-buffered, so this instruction does not actually take effect until the next display field occurs. This avoids the possibility of some routine trying to update the Copper instruction list while the Copper is trying to use it to create the display.

Synchronizing with the Display

To synchronize your routines with the display, you use a call to the system routine **WaitTOF()**. Although your routines may possibly be capable of generating more than 60 complete display fields per second, the system itself is limited to 60 displays per second. Therefore, after generating a complete display, you may wish to wait until that display is ready to be shown on the screen before starting to generate the next one. **WaitTOF()** holds your task until the vertical-blanking interval (blank area at the top of the screen) has begun. At that time, the system has retrieved the current Copper instruction list and is ready to allow generation of a new list.

The call to the vertical-blanking synchronization routine takes the following form:

```
WaitTOF();
```

Now that you have learned how to add and display **VSprites**, you may want to change some of their characteristics, as shown in the following section.

Changing VSprites

Once the **VSprite** has been added to the GEL list and is in the display, you can change some of its characteristics with the following operations:

- o Pointing to a new **VSprite** image (change the **ImageData** pointer)
- o Pointing to a new **VSprite** color set (change the **SprColors** pointer)
- o Defining a new **VSprite** position (change the y,x values)

VSPRITE OPERATIONS SUMMARY

This section provides a summary of the **VSprite** operations in their proper sequence:

- o Define a **View** structure that you can later merge with the **VSprite** instructions.
- o Initialize the GEL system (call **InitGels()**). This only needs to be done once.
- o Define the **VSprite**:
 - Define height.

- Define on-screen position.
- Define where to find **ImageData** data.
- Define where to find **SprColors** to use.
- Define **VSprite** structure flags to show that this is a **VSprite**.
- o Add the **VSprite** to the GEL list.
- o Change the **VSprite** appearance by doing the following:
 - Changing the pointer to **ImageData**.
 - Changing its height.
- o Change the **VSprite** colors by changing the pointer to **SprColors**.
- o Move the **VSprite** by defining a new y,x position.
- o Display the **VSprite** with this sequence of routines:
 - **ON_DISPLAY;**
 - **ON_SPRITE;**
 - **SortGList()**
 - **DrawGList()**
 - **MrgCop()**
 - **LoadView()**

Once you have mastered the basics of handling **VSprites**, you may want to study the next two sections to find out how to reserve hardware sprites for use outside the **VSprite** system and how to assign the **VSprites**.

VSPRITE ADVANCED TOPICS

This section describes advanced topics pertaining to **VSprites**. It contains details about reserving hardware sprites for use outside of the **VSprite** system, information about how **VSprites** are assigned, and more information about **VSprite** colors.

Reserving Hardware Sprites

To prevent the **VSprite** system from using specific hardware sprites, you can write into the variable named **sprRsrvd** in the **GelsInfo** structure. The pointer to the **GelsInfo** structure is contained in the **RastPort** structure. If the contents of this 8-bit value is zero, then all of the hardware sprites may be used by the **VSprite** system. If any of the bits is a 1, the sprite corresponding to that bit will not be utilized by **VSprites**. Note that this increases the likelihood of a **VSprite** **VSOVERFLOW**. See the next section, "How **VSprites** are Assigned," for further details on this topic.

Hardware sprites are reserved as shown below.

This sprite is reserved: 7 6 5 4 3 2 1 0

If this **sprRsrvd** bit is a 1: 7 6 5 4 3 2 1 0

You normally assign hardware sprites in pairs, as suggested by the following example. Suppose you want to reserve sprites 0 and 1. Your program would typically include the following kinds of statements:

```
struct RastPort myRastPort; /* the View structure is defined */
...
...
myRastPort->GelsInfo->sprRsrvd = 0x03; /* reserve 0 and 1 */
```

If you reserve a hardware sprite for your own use, the system is unable to use that hardware sprite when it makes a **VSprite** assignment. In addition, because pairs of hardware sprites share color register sets, reserving one hardware sprite effectively eliminates two.

If you are using the simple sprite system to allocate sprites, you can look in the **GfxBase** structure to see which sprites are already in use.

Note: If **Intuition** is running, sprite 0 is already reserved for use as the pointer.

The reserved sprite status is accessible as

currentreserved = GfxBase->SpriteReserved

The next section presents a few trouble-shooting techniques for **VSprite** assignment.

How VSprites Are Assigned

Each **VSprite** can display three possible colors plus transparent. To define colors for **VSprites**, you use the **SprColors** pointer. **SprColors** points to the first of three word quantities, representing the three possible pixel colors for that virtual sprite.

Although the **VSprites** are handled by the automatic routines, the system *may* run out of sprites. If you ask that the software display more than four **VSprites** on a single horizontal scan line, it is possible that one or more sprites may disappear until the conflict is resolved.

Here is the reason that the **VSprite** routines might have problems, and some suggestions on how to avoid them. There are 8 *real* sprite DMA channels. Sprites 0 and 1 share color registers 17-19; sprites 2 and 3 share registers 21-23; sprites 4 and 5 share registers 25-27; and sprites 6 and 7 share registers 29-31.

When the **VSprite** routines use the sorted list of **VSprite** elements, they build a Copper instruction list that decides when to reuse a sprite DMA channel. They also build a Copper instruction stream that stuffs the color register set for the sprite selected at that time on the screen to represent this **VSprite** image.

This process consists of the following steps:

1. Use real sprite 0 to represent the first virtual sprite. Load that virtual sprite's colors into the three color registers for sprite 0 (registers 17, 18, 19).
2. Now look at the rest of the virtual sprites the user wishes to display on this same horizontal line.
3. If the **VSprite** color pointers are all different from the pointer found in the sprite 0 pointer, it will not be possible to use the real sprite 1 DMA channel for display on this line because it shares the real sprite 0 colors.
4. Conversely, if one of the other virtual sprites to appear on this line shares the same virtual color pointer, the **VSprite** routines can use sprite DMA channel 1 to represent that second virtual sprite.

5. The **VSprite** routines continue to map virtual sprites against the real sprites until either of the following events occurs:
 - o All virtual sprites are assigned.
 - o The system runs out of real sprites that it can use.

The system will run out of real sprites to use if you ask the virtual sprite system to display more than four sprites having different pointers to their color table on the same horizontal line. During the time that there is a conflict, one or more of your virtual sprites will disappear.

You can avoid these problems by taking the following precautions:

- o Minimize the number of **VSprites** you wish to appear on a single horizontal line.
- o If colors for some virtual sprites are the same, make sure that the pointer for each of the **VSprite** structures for these virtual sprites points to the same memory location, rather than to a duplicate set of colors elsewhere in memory.

If You Do Not Specify VSprite Colors

To pick the set of colors to use, you specify the pointer named **SprColors**. If you specify a 0 value for **SprColors**, that **VSprite** does *not* generate a color-change instruction stream for the Copper when the system displays it. Instead, the **VSprite** appears drawn in the color set that is currently written into the color registers for the hardware sprite currently selected to display this **VSprite**.

Table 3-2 shows how the hardware sprites use the color registers to select their possible range of colors:

Table 3-2: Hardware Sprite Color Registers

Hardware Sprite	Color Registers
-----------------	-----------------

0 and 1	17 - 19
2 and 3	21 - 23
4 and 5	25 - 27
6 and 7	29 - 31

During one screen display, the system may use hardware sprite number 1 to display a **VSprite**. In this case, the **VSprite** selects its three available colors from color register numbers 17-19. On another screen display, the system may select hardware sprite number 7 to display the same

VSprite. In this case, the hardware sprite uses color registers 29-31.

Therefore, if you make the **SprColors** pointer a 0, specifying that color does not matter, the system may display your **VSprite** in any one of a set of four different possible color groupings as indicated in the table above.

How VSprite and Playfield Colors Interact

The **VSprites** use system color registers 16 through 31 to hold the **VSprite** color selections. There are only 32 color registers in the system. The highest 16 color registers (16-31) are shared with the playfield color selections. If you are working in 32-color low-resolution mode, the system makes the first 16 color selections for the playfield pixels from color registers 0-15 and then makes the remaining color selections from color registers 16-31.

If you are using the **VSprite** system and specifying the colors (using **SprColors**) for each **VSprite**, the contents of color registers 16-31 will change constantly as the video display beam progresses down the screen. The Copper instructions change the registers to display the correct set of colors for your **VSprites** depending on their positions. If you have any part of a 32-color playfield display drawn in any of the colors shown in table 3-2, those colors will appear to flicker and change as your **VSprites** move.

This problem also affects 32-color **Bobs** because **Bobs** are actually drawn as part of the playfield display. Anything that affects the playfield affects the **Bobs** as well.

You can avoid this flickering and changing of colors by taking the following precautions:

- o Use no more than 16 colors in the playfield display whenever you use **VSprites**; or
- o If you are using a 32-color playfield display, do not use any colors other than 0-15, 16, 20, 24, and 28. The remaining color numbers are used by the **VSprite** system; or
- o Specify the **VSprite SprColors** pointer as a value of 0. This avoids changing the contents of any of the hardware sprite color registers, but may cause the **VSprites** to change colors depending on their positions relative to each other, as described in the previous section.

The first two alternatives are the easiest to implement.

Using Bobs

Because **Bobs** and **VSprites** are both graphics objects handled by the GEL system, they share many of the same data requirements. **VSprites** and **Bobs** differ primarily in that **Bobs** are drawn into the playfield using the blitter, while **VSprites** are assigned to hardware sprites.

The following sections describe how to define a **Bob**, including how to specify its size, select its colors, form its image, and specify its on-screen position.

Because a **Bob** is a more complex object than a **VSprite**, you must also define various other items, such as the color depth of the **Bob**, how to handle the drawing of the **Bob**, and certain other variables that the GEL system requires when **Bobs** are used.

LINKING A BOB TO A VSPRITE STRUCTURE

To fully define a **Bob**, you define two different structures: a **VSprite** structure and a **Bob** structure. The graphics animation system has been designed as a set of interrelated elements, each of which builds on the information provided by the underlying structure to create additional versatility. The common elements—such as height, collision-handling information, position in the drawing area, and pointers to the data definition—are part of the **VSprite** structure. The added features—such as drawing sequence, data about saving and restoring the background, and other features not common to **VSprites**—are part of the **Bob** structure instead.

The **VSprite** and **Bob** structures must point to one another, so that the system knows where all of the appropriate variables are defined. For example, suppose your program defines two structures that are to define a **Bob** named “myBob” as follows:

```
struct Bob myBob;  
struct VSprite myVSprite;
```

You must create a link between the two structures with a set of program statements such as:

```
myBob.BobVSprite = &myVSprite;  
myVSprite.VSBob = &myBob;
```

Now the system can go back and forth between the two structures to obtain the various elements as needed to define the **Bob**.

SPECIFYING THE SIZE OF A BOB

Whereas a **VSprite** was limited to 16 pixels of width, a **Bob** can be any size you wish to define. To specify the size of a **Bob**, you use not only the **Height** but also the **Width** variable. You specify these variables in a **VSprite** structure associated with the **Bob**. Specify the width as the number of 16-bit words it takes to fully contain the object.

As an example, suppose the **Bob** is 24 pixels wide and 20 lines tall. You use statements such as the following to specify the size:

```
myVSprite.Height = 20; /* 20 lines tall */  
myVSprite.Width  = 2; /* 2 words = 24 pixels wide, rounded  
                       * up to the next multiple of 16 pixels. */
```

Because **Bobs** are drawn into the playfield background, the pixels of the **Bob** are the same size as the background pixels. With hardware sprites, the pixels are of a fixed size (low-resolution pixels).

SPECIFYING THE COLORS OF A BOB

Because a **Bob** is drawn into the playfield area, it can have as many colors as the playfield area itself. Typically a five-bit-plane, low-resolution mode display allows you to select playfield pixels (and therefore, **Bob** pixels) from any of 32 active colors out of a system palette of 4,096 different color choices. The set of colors you select for the playfield area is the set of colors the system uses to display the **Bobs**.

For **Bobs**, the system ignores the **SprColors** variable in the **VSprite** structure. You use the **Depth** variable in the **VSprite** structure to define how much data is provided to define the **Bob**. This variable also defines how many different colors you can choose for each of the pixels of a **Bob**.

The **Depth** variable specifies how many bit-plane images the system must retrieve from the **Bob** image data area to make up the **Bob**. These are called bit-plane images as the system will write each image into a different bit-plane. The combination of bits in identical y,x positions in each bit-plane determines the color of the pixel at that position.

For example, if you specify only one plane, then the bits of that image let you select only two different colors: one color for each bit that is a 0, a second color for each bit that is a 1. Likewise, if there are 5 images stored sequentially and you specify a depth of 5, each image contributes one bit for each position in the image to the color number selector, allowing up to 32 different choices of color for each **Bob** pixel.

You specify depth using a statement such as the following:

```
myVSprite.Depth = 5; /* allow 32 colors; requires that a
                      * 5-bit-plane image be present in data area. */
```

SPECIFYING THE SHAPE OF A BOB

The organization of a **Bob** in memory is different from that of a **VSprite** because of the way the system retrieves data to draw **Bobs**. To define a **Bob**, you must still initialize the **ImageData** pointer to point to the first word of the image definition; however, the layout of the data is different for **Bobs** than for **VSprites**.

The sample image below shows the same image defined as a **VSprite** in the “Using VSprites” section above. The data, however, is stored in a way typical of a **Bob**.

If a shape is 2 bits “deep” and is a triangular shape, you would lay it out in memory as follows:

```
          <first bit-plane data>
mem      1111111111111111
mem + 1  0011000000001100
mem + 2  0000111111111000
mem + 3  0000001111000000
mem + 4  0000000110000000

          <second bit-plane data>
mem + 5  1111111111111111
mem + 6  0011111111111100
mem + 7  0000110000110000
mem + 8  0000001111000000
mem + 9  0000000110000000

          <<third bit-plane data>
          ...
          <<fourth bit-plane data>
          ...
          <<fifth bit-plane data>
          ...
```

To state the width of the **Bob** image, you use 16-bit words. The **Width** value is the number of words that fully contain the image. For example, you store a 29-bit wide image in 32 bits (2 data words of 16 bits each) for each line of its data.

You still specify the number of lines with the **Height** variable in the **VSprite** data structure. However, you treat **Height** somewhat differently for a **Bob** than for a **VSprite**. Specifically, for a **VSprite**, two adjacent data words that always occur together define the colors of each **VSprite** pixel. For a **Bob**, the **Height** variable defines how many adjacent data words it takes to define one complete bit-plane image. That is, for a **Bob** the number of adjacent data words in each bit-plane image definition is given by the following formula: **Height x Width**.

The **Depth** variable defines how many adjacent (end-to-end) images there are in the data area to define the shape of the **Bob**. See the example at the end of the "PlaneOnOff" section below.

OTHER ITEMS INFLUENCING BOB COLORS

Three other variables in the **VSprite** structure affect the color of **Bob** pixels: **PlanePick**, **ImageShadow**, and **PlaneOnoff**.

PlanePick

Assume that you have defined a playfield composed of five bit-planes. The variable **PlanePick** in the **VSprite** structure lets you specify which of the bit-planes are to be affected when the system draws the **Bob**. **PlanePick** binary values affect the bit-planes according to the following pattern:

Draw Bob into this bit-plane:	5 4 3 2 1 0
If this PlanePick bit is a 1:	5 4 3 2 1 0

For example, if **PlanePick** has a binary value of:

0 0 0 1 1

then the system draws the first bit-plane image of the **Bob** into bit-plane 0 and the second image into bit-plane 1.

Suppose that you still want to define an image of only 2 bit-planes, but wish to draw the **Bob** into bit-planes 1 and 4 instead of 0 and 1. Simply choose a **PlanePick** value of:

1 0 0 1 0

This value means "write first image into plane 1, second image into plane 4."

ImageShadow

The variable named **ImageShadow** is a pointer to a memory area that you have reserved for holding the shadow mask of a **Bob**. A shadow mask is the logical *or* combination of all 1-bits of a **Bob** image. There is a variable in the **VSprite** structure called **CollMask** (pointer to a collision mask, covered under “Topics Common to Both VSprites and Bobs”) for which you reserve some memory space. The **ImageShadow** and **CollMask** pointers usually, but not necessarily, point to the same data.

Figure 3-2 shows an example of a shadow mask with only the 1 bits.

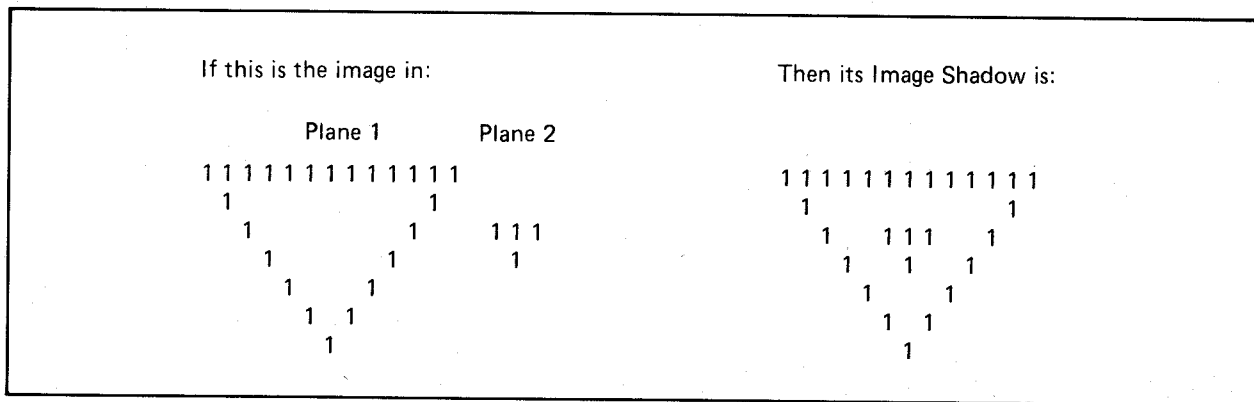


Figure 3-2: An Image and Its ImageShadow

The system uses the shadow mask along with the variable **PlaneOnOff**, discussed in the next section. Because **ImageShadow** in the **Bob** structure is a pointer to a data area containing the sprite shadow, you must provide space that the the system can use for this purpose. You must then initialize the pointer to the first location within the data area that you have set aside. You can calculate the minimum size of this area as follows:

$$\text{shadow size} = \text{Height} * \text{Width}$$

So, for example, an object 5 lines high by 32 bits wide (**VSprite** or **Bob**) requires a sprite shadow storage area of at least 5 x 32, or ten 16-bit locations. The example in the “PlaneOnOff” section below shows how to reserve the memory for the sprite shadow and how to tell the system where to find it.

PlaneOnOff

The variable named **PlaneOnOff** tells the system what to do with the playfields that are not “picked” (affected) by **PlanePick**. The binary bit positions for **PlaneOnOff** are the same as those for **PlanePick** (lowest bit position specifies the lowest-numbered bit-plane). However, their meaning differs. For every plane position *not* selected by **PlanePick**, parts of the non-selected plane are filled with the value shown in the corresponding position of **PlaneOnOff**. The parts that are filled are the positions where there is a 1-bit present in the sprite’s image shadow.

This provides a great deal of versatility. You can use a two-plane **VSprite** image as the source for many **Bob** images. Yet, because of the color combinations each contains, it may seem that there are several different images present.

For example, assume that the data shown in the **Bob** layout above defines a two-bit-plane **Bob** image that selects its colors from color registers 0, 1, 4, and 5. To initialize the **Bob** and **VSprite** structures, you need to provide the following types of statements:

```
/* data definition from example layout */
WORD BobData[] = {
    0xFFFF, 0x300C, 0x0FF0, 0x03C0, 0x0180,
    0xFFFF, 0x3FFC, 0x0C30, 0x03C0, 0x0180
};

/* reserve space for the collision mask for this Bob */
WORD BobCollision[10];

myVSprite.Width = 1; /* sample image is 16 pixels wide (1 word) */
myVSprite.Height = 5; /* takes 5 lines to define each image of the Bob */
myVSprite.Depth = 2; /* only two bit-plane images are defined in BobData */

/* show the system where it can find the data image of the Bob */
myVSprite.ImageData = BobData;

/* binary = 00101, means draw into only bit-planes 0 and 2 */
myVSprite.PlanePick = 0x05;

/* binary = 00000, means for planes not picked, that is, 1, 3, and 4,
 * fill those planes with 0's wherever there is a 1 in the sprite shadow mask
 */
myVSprite.PlaneOnOff = 0x00;

/* where to put collision mask */
myVSprite.CollMask = BobCollision;
```

```

/* tell the system where it can assemble a sprite shadow */
/* point to same area as CollMask */
myBob.ImageShadow = BobCollision;

/* create the sprite collision mask for this Bob's VSprite structure */
InitMasks( &myVSprite );

```

Whenever the system draws this **Bob**, it fills any position where there is a 1 in the sprite shadow with a 0 for any plane not selected by **PlanePick**. Therefore, the only binary combinations the **Bob** pixels can form are as shown below. Because of **PlanePick**, 1s can appear only at these two locations: 0 0 1 0 1. So the color choices are limited to the following:

Color Selected	Binary Combination
color 0	0 0 0 0 0
color 1	0 0 0 0 1
color 4	0 0 1 0 0
color 5	0 0 1 0 1

These color choices fulfill the requirements specified for the example.

To select the position of a **Bob**, specify the y and x variables in the **VSprite** structure associated with the **Bob**. For example:

```

myVSprite.Y = 100;
myVSprite.X = 100;

```

BOB PRIORITIES

This section describes the two choices you have for system priorities between **Bobs**. You can ignore the priority issue and let the system decide which **Bob** has the highest priority, or you can specify the drawing order yourself. When you specify the drawing order, you control which **Bob** the system draws *last*, and therefore, which one appears in front of other **Bobs**.

Letting the System Decide Priorities

If you want the system to decide, you set the **Before** and **After** pointers in the **Bob** data structure to zero. In this case, the system draws the **Bobs** in their y,x positional order on the screen. In other words, the system draws whichever object is on the screen and is currently the highest within the drawing area (lowest y coordinate value). If two objects have the same y coordinate, the object that has the lowest x coordinate value is drawn first.

The **Bob** drawn first has the lowest priority. The **Bob** drawn last has the highest priority because later objects overlap the objects drawn earlier.

As you use the animation system to move objects past each other on the screen, you will notice that sometimes the objects switch priorities as they pass each other. For example, suppose you want the system to establish the priorities of the **Bobs**, and there are two **Bobs** defined in the system—myBob2 and myBob3. You set the **Before** and **After** pointers as follows:

```
myBob2.Before = 0;
myBob2.After  = 0;
myBob3.Before = 0;
myBob3.After  = 0;
```

Specifying the Drawing Order

If you wish to specify the priorities, simply specify the pointers as follows. **Before** points to the **Bob** that this **Bob** should be drawn before, and **After** points to the **Bob** that this **Bob** should be drawn after. This guarantees that **Bob** objects retain their relative priorities.

For example, suppose you want to assure that myBob3 always appears in front of myBob2. You must initialize the **Before** and **After** pointers so that the system will always draw myBob3 last; that is, after myBob2.

```
myBob2.Before = &myBob3; /* draw Bob2 before drawing Bob3 */
myBob2.After  = 0;        /* draw Bob2 after no other Bob */
myBob3.After  = &myBob2; /* draw Bob3 after drawing Bob2 */
myBob3.Before = 0;        /* draw Bob3 before no other Bob */
/* draw nothing in particular after this Bob */
```

If you decide to specify the **Before** and **After** pointers for any one **Bob** in a group, then you must also at least set the **Before** and **After** pointers to zero for *all* of the rest of the **Bobs** in that group.

For example, if there are ten **Bobs** and you only care that the system draws numbers 4, 6, and 9 in that sequence, you must properly fill in the **Before** and **After** pointers for these three **Bobs**. If you do not care in which order the system draws the other seven **Bobs**, you need only initialize their **Before** and **After** pointers to a value of 0 to assure correct treatment by the system.

You must properly point *all* **Before** and **After** pointers of a group to each other because the **Bob** that is the upper-leftmost becomes the first the system considers for drawing. The system follows the **Before** pointers until it finds one having a zero value, and draws that **Bob** first. It then draws other **Bobs** in the sequence you have specified.

In the example code sequence above, the comment “draw nothing in particular after this Bob” simply means that once the drawing sequence for this set of **Bobs** has been performed, the system still proceeds to find and draw all other **Bobs** currently linked into the GEL list. To continue the drawing operation, the system simply goes on searching the list for the next **Bob** whose **Before** pointer is 0.

Specifying Priority between Bobs and VSprites

See “Topics Common to Both VSprites and Bobs” below for details.

SAVING THE PLAYFIELD DISPLAY

Once the system has drawn the **Bobs**, they become part of the playfield segment of the display. The image of a **Bob** overlays part of the background area. To move a **Bob** from one place to another, you must tell the system to save the background before it draws the **Bob** and to restore the background to its original condition when it moves the **Bob**.

A variable called **sprFlag** in the **VSprite** structure contains a flag called **SAVEBACK**. To cause the system to save and restore the background for that **Bob**, set the **SAVEBACK** flag to 1.

In addition to the **sprFlag** variable, you must also tell the system where it can put this saved background area. For this, you use the **SaveBuffer** variable. For example, if the **Bob** is 48 pixels wide and 20 lines high, and the system is drawing it into a playfield of five bit-planes, you must allocate space for storing the following:

$$(48 \text{ pixels}/16 \text{ pixels per word}) * (20 \text{ lines}) * (5 \text{ bit-planes}) = 300 \text{ words}$$

To allocate this space, use the graphics function **AllocRaster()**. When you use **AllocRaster()** for this purpose, you can specify the area size in bits, so it may well be the most convenient way to reserve the space you need. For example:

```
myBob.SaveBuffer = AllocRaster(48,20 * 5);
/* save space to store 48 bits times 20 words times 5 bit-planes */
```

Note that the **AllocRaster()** function rounds the width value up to the next integer multiple of 16 bits.

USING BOB FLAGS

The following sections describe the **Bob** flags. Some of these are in the **VSprite** structure associated with the **Bob**; others are in the **Bob** structure itself. The description of each flag tells the structure in which the flag is located.

VSPRITE Flag

If you are using the **VSprite** structure to describe a **Bob**, set **VSPRITE** to zero.

The **VSPRITE** flag is located in the **VSprite** structure.

SAVEBACK Flag

If you want the GEL routines to save the background before the **Bob** is drawn and to restore the background after the **Bob** is removed, set the **SAVEBACK** (for “save the background”) flag in the **VSprite** structure to 1.

If you set this flag, you must have allocated the buffer named **SaveBuffer**.

OVERLAY Flag

If the system should use the sprite shadow mask when it draws the **Bob** into the background, set the **OVERLAY** flag in the **VSprite** structure to 1. If this flag is set, it means that the background original colors show through in any section where there are 0 bits in the sprite shadow mask. Essentially, then, those 0 bits define areas of the **Bob** that are “transparent.”

If you set the **OVERLAY** bit to a value of 0, the system uses the *entire rectangle* of words that define the **Bob** image and uses its contents to *replace* the playfield area at the specified y,x coordinates.

If you set this flag, you must have allocated space for and initialized the **ImageShadow** shadow mask. See the section above called “Sprite Shadow Mask” for details on the shadow mask.

GELGONE Flag

The system sets this flag in the **VSprite** structure to indicate when the **Bob** has been moved to y,x coordinates entirely outside of the "clipping region."

When an object crosses over certain specified boundaries in the drawing area, the system does not draw all of the object into the background but "clips" (truncates) it to those limits. At the time of this writing, the variables named **topmost**, **bottommost**, **leftmost**, and **rightmost** define the minimum and maximum y,x coordinates of this clipping region.

When the system sets the GELGONE flag to a 1, you know that the object has passed entirely beyond those limits and that the system will not draw any part of the object into the drawing area. On the basis of that information, you may decide that the object need no longer be part of the GEL list and may decide to remove it to speed up the consideration of other objects.

SAVEBOB Flag

To tell the system not to erase the old image of the **Bob** when the **Bob** is moved, set the SAVEBOB flag in the **Bob** structure to 1. This lets you use the **Bob** like a paintbrush if you wish. It has the opposite effect of SAVEBACK.

Note: It takes longer to preserve and restore the raster image than simply to draw a new **Bob** image wherever required.

BOBISCOMP Flag

If this **Bob** is part of an **AnimComp**, set the BOBISCOMP flag in the **Bob** structure to 1. If the flag is a 1, you must also initialize the pointer named **BobComp**. Otherwise, the system ignores the pointer, and it may be left alone. See "Animation Structures and Controls" for a discussion of **AnimComps**.

BWAITING Flag

When a **Bob** is waiting to be drawn, the system sets the BWAITING flag in the **Bob** structure to 1. This occurs only if the system has found a **Before** pointer in this **Bob's** structure that points to another **Bob**. Thus, the system flag BWAITING provides current draw-status to the system. Currently, the system clears this flag on return from each call to **DrawGList()**.

BDRAWN Flag

The **BDRAWN** system status flag in the **Bob** structure tells the system that this **Bob** has already been drawn. Therefore, in the process of examining the various **Before** and **After** flags, the drawing routines may determine the drawing sequence. Currently, the system clears this flag on return from each call to **DrawGList()**.

BOBSAWAY Flag

To initiate the removal of a **Bob** during the next call to **DrawGList()**, set **BOBSAWAY** to 1. Either you or the system may set this **Bob** structure system flag. The system restores the background where it has last drawn the **Bob**. The system will unlink the **Bob** from the system GEL list the next time **DrawGList()** is called unless you are using double-buffering. In that case, the **Bob** will not be unlinked and completely removed until two calls to **DrawGList()** have occurred and the **Bob** has been removed from both buffers.

BOBNIX Flag

When a **Bob** has been completely removed, the system sets the **BOBNIX** flag to 1 on return from **DrawGList()**. In other words, when the background area has been fully restored and the **Bob** has been removed from the GEL list, this flag in the removed **Bob** is set to a 1. **BOBNIX** is significant when you use double-buffering, because once you ask that a **Bob** be removed, the system must remove it from the active drawing buffer and from the display buffer. Once **BOBNIX** has been set for a double-buffered **Bob**, it has been removed from both buffers and you are free to reuse it or deallocate it.

This flag is in the **Bob** structure.

SAVEPRESERVE Flag

The **SAVEPRESERVE** flag is a double-buffer version of the **SAVEBACK** flag. If you are using double-buffering and wish to save and restore the background, you set **SAVEBACK** to 1. **SAVEPRESERVE** is used by the system to indicate whether the **Bob** in the “other” buffer has been restored; it is for system use only.

ADDING A BOB

To add a **Bob** to the system GEL list (the same list you created for **VSprites** using **InitGels()**), you use the **AddBob()** routine. It is advisable that you initialize the different variables you plan to use within the **Bob** structure before you ask that the system add this **Bob** to the list.

For example:

```
struct GelsInfo myGelsInfo;
struct VSprite dummySpriteA, dummySpriteB;
struct Bob myBob;

/* done ONCE, for this GelsInfo */
InitGels( &dummySpriteA, &dummySpriteB, &myGelsInfo );

/* here initialize the Bob variables */
AddBob( &myBob, &rastport );
```

REMOVING A BOB

Two methods may be used to remove a **Bob**. This section describes the system routine for each method.

The first method uses the **RemBob()** routine. You call this routine as follows:

```
RemBob ( &myBob, &rastport );
```

RemBob() causes the system to remove the **Bob** during the next call to **DrawGList()** (or two calls to **DrawGList()** if the system is double-buffered). **RemBob()** asks the system to remove the **Bob** "at its next convenience."

The second method uses the **RemIBob()** routine. For example:

```
RemIBob ( &myBob, &rastport, &viewport );
```

RemIBob() tells the system "remove this **Bob** immediately!" It causes the system to erase the **Bob** from the drawing area and causes the immediate erasure of any other **Bob** that had been drawn subsequent to this one. The system then unlinks the **Bob** from the system GEL list. To redraw the **Bobs** that were drawn on top of the one just removed, you must make another call to **DrawGList()**.

GETTING THE LIST OF BOBS IN ORDER

Like the list of **VSprites**, the list of **GELS** must be in the proper y,x sorted order from top of screen to bottom and from left to right. The system uses the position information to decide drawing sequences if you have not specified otherwise by using the **Before** and **After** pointers. You must therefore assure that the **GEL** list is sorted before you ask the system to display the **Bobs**.

To sort the **GEL** list, you call **SortGList()**. For example:

```
SortGList( &rastport );
```

DISPLAYING BOBS

This section provides the typical sequence of operations for drawing the **Bobs** on the screen. It is very similar to that shown for **VSprites**, as both **Bobs** and **VSprites** are **GELS** and are part of the same list of controlled objects.

Specifically, the system automatically synchronizes the drawing routines to the display beam and may not require that the display be turned off during the update. If large **Bobs** or many **Bobs** are created, you may be interested in double-buffering. See the section called “Double-Buffering” in this chapter for details.

When you call **DrawGList()**, the system *actually draws* any **Bobs** on this list into the area you have specified. The system saves the backgrounds if you have provided for the save and then performs the drawing sequence in the order you requested. To initiate this drawing, call **DrawGList()**. For example:

```
struct RastPort *rp;  
struct ViewPort *vp;  
...  
DrawGList(rp, vp); /* draw the elements */
```

CHANGING BOBS

You can change the following characteristics of **Bobs**:

- o To change their appearance, change the pointer to the **ImageData** in the associated **VSprite** structure. Note that the change in the **ImageData** pointer also requires a change in the **ImageShadow** or a recalculation of the object mask, using **InitMasks()**.

- o To change their color choices, change their **PlanePick** and/or **PlaneOnOff** values; also change the depth parameters if the sprite image has multiple planes defined.
- o To change the location in the drawing area, change the y,x values in the associated **VSprite** structure.
- o To change the object priorities, change the drawing sequence by altering the **Before** and **After** flags in the **Bob** structures.
- o To change the **Bob** into a paintbrush, set the **SAVEBOB** flag to a 1 in the **Bob** structure.

Note: Neither these nor other changes actually happen until you call **SortGList()** and then **DrawGList()**.

DOUBLE-BUFFERING

Double-buffering is the technique of supplying two different memory areas in which the drawing routines may create images. The system displays one memory space while you are drawing into the other area. This assures that you never see any display fields on the screen that consist partly of old material and partly of new material.

The system animation routines use an extension that you establish to the **Bob** structure. Also, if you do not care to use double-buffering, you need not tie up precious memory resources for unneeded variable storage space.

To find whether a **Bob** is to be double-buffered, the system examines the pointer named **DBuffer** in the **Bob** structure. If this pointer has a value of 0, the system does not use double-buffering for this **Bob**.

Note: If you do *not* wish to use double-buffering, you must initialize the **DBuffer** pointer to zero. For example:

```
myBob.DBuffer = 0;    /* do this if this Bob is NOT double-buffered */
```

The next section discusses several other variables that you must describe if you want to use double-buffering. *Note:* if any of the **Bobs** are double-buffered, then *all* of them must be double-buffered.

Variables Used in Double-Buffering

To use double-buffering for a given **Bob**, you must provide a data packet for the system to store some of the variables it needs to handle double-buffering. This data packet is a structure named **DBufPacket** that consists of the following variables:

BufY, BufX

System variables that let the system keep track of where the object was located “last screen” (as compared to the **Bob** structure variables called **oldY** and **oldX** that tell where the object was two screens ago). **BufY** and **BufX** provide for correct restoration of the background within the currently active drawing buffer.

BufPath

System variable related to the drawing order used to draw this **Bob** into the background. **BufPath** assures that the system restores the backgrounds in the correct sequence; it relates to the system variables **DrawPath** and **ClearPath** (found in this **Bob**'s **VSprite** structure).

BufBuffer

You must set this field to point to a buffer as big as this **Bob**'s **SaveBuffer** to allocate separate space for buffering the background on which you are drawing the **Bob**. This buffer is used to store the background for later restoration when the system moves the object.

The next section shows how to pull all these variables together to make a double-buffered **Bob**.

Creating a Double-Buffered Bob

To create a double-buffered **Bob**, you must initialize all of the normal **Bob** variables and pointers and execute a code sequence similar to the following:

```
struct DBufPacket myDBufPacket;

/* allocate a DBufPacket for myBob */
...
...
/* same size as previous example in “Saving the Playfield Display” */
myDBufPacket.BufBuffer = AllocRaster( 48, 20 * 5 );

/* tell Bob about its double buff status */
myBob.DBuff = myDBufPacket;
```

BOB OPERATIONS SUMMARY

The following steps are involved in defining, moving, and displaying a **Bob**:

- o Define a **RastPort** structure for the drawing routine to use.
- o Initialize the GEL system (call **InitGels()**) for this **RastPort**. You only need to do this once.
- o Create and link a **Bob** and a **VSprite** structure.
- o Define the following **Bob** parameters:
 - Height
 - Width
 - Depth
 - Position
 - Where to find **ImageData** data
 - Which planes to pick for writing this **Bob**
 - How to treat the planes not picked
 - **VSprite** structure flags to show that this is a **Bob**
 - Space for the sprite shadow
 - Pointer to a **DBufPacket** if you want to use double-buffering (otherwise, make this pointer a NULL (0) value)
- o Call **InitMasks()** to create the sprite shadow.
- o Add the **Bob** to the GEL list.
- o Change the **Bob** appearance by
 - Changing the pointer to **ImageData**
 - Changing its height, width or depth

- o Change the **Bob** colors by
 - Changing the playfield color set
 - Changing **PlanePick** and **PlaneOnOff**
- o Move the **Bob** by defining a new y,x position.
- o Display the **Bob** by calling:
 - **SortGList()**;
 - **DrawGList()**;

Now that you've mastered the basics of handling **VSprites** and **Bobs**, you may want to find out about some of the interactions between the two and how to cope with these interactions. Or, you may want to skip these advanced topics and read about software collisions, clipping, and adding new features in "VSprite and Bob Topics" below.

BOB ADVANCED TOPICS

How Bob Colors Are Controlled

Bobs do not use the **SprColor** pointer. To determine the color of a **Bob**, you use the existing colors in the 32-entry color table. The lower 16 of the 32 possible color selections (registers 0-15) are always dedicated to playfield color selections, providing 16 unique colors for the **Bobs**, since they are playfield objects.

However, the playfields and the **VSprites** share the upper 16 of the 32 color entries (registers 16-31). If you are using five bit-planes to display the **Bobs**, any **Bob** with a pixel whose color value exceeds 15 may change color if the virtual sprites are running at the same time.

Note: This also applies to *any* static part of the display area (the playfield), whether a **Bob** or simply part of the background display, for which a five- or six-bit-plane image is used if the color number for a specific pixel exceeds the value of 15.

To explain further, the virtual sprite routines, notably **SortGList()** and **DrawGList()**, work together to decide which real sprite will be used at any point on the screen. **DrawGList()** makes up a Copper instruction list to change the contents of the upper 16 color registers, perhaps several times within a single display field. Therefore, depending on where a **Bob** image is on the screen relative to a virtual sprite, and depending on its color content, a **Bob** may take on different colors (perhaps even within only a part of its body).

To minimize color interactions between **Bobs** and virtual sprites, take the appropriate precautions:

- o Limit the background to four or fewer bit-planes and thus limit the **Bob** color choices to 16 or fewer.
- o Use five bit-planes, but specify **Bob** colors or background colors from the colors 0 through 15 or 16, 20, 24, or 28 only. Colors 16, 20, 24, and 28 are used neither by real sprites nor by virtual sprites and are treated as transparent areas. Therefore, if you use only these colors for **Bobs**, the simultaneous use of virtual sprites will not affect the **Bob** or background colors.
- o Use **sprRsrvd** to “fence-off” certain sprite pairs, so you can also use their colors for **Bobs**.

Topics Common to Both VSprites and Bobs

DETECTING GEL COLLISIONS

To detect collisions between graphics elements, you use the **DoCollision()** routine. **DoCollision()** determines if there are any pixels of one graphics element currently touching those of another graphics element or if any of the graphics elements have passed outside of specified screen boundaries.

Whenever there is a collision, the system performs one of 16 possible collision routines. The addresses of the collision routines are kept in a table called the collision handler table. **DoCollision()** examines the **HitMask** and **MeMask** of each of the **VSprite** structures in the GEL list and determines if there is a collision between any two GELS. It then calls the collision-handler routine at the table position corresponding to the bits in the **HitMask** and **MeMask**, as outlined below.

Note: The current form of these routines does *not* use the built-in *hardware* collision detection. You may, if you wish, reserve one or more sprites for your own use and move them using your own routines. When specific sprites have been reserved for your own use, you may choose to use the hardware collision detection to sense collisions between your own objects and other on-screen elements. See the *Amiga Hardware Reference Manual* for information about hardware collision detection.

Default Kinds of Collisions

Two kinds of software collisions are handled by the collision routines: boundary hits and GEL-to-GEL hits.

You can set up routines to handle as many as 16 different kinds of collisions using the **VSprite** structure **MeMask** and **HitMask**. When you call a collision routine, you give it certain kinds of information about the colliding elements, as described in the next two sections.

Boundary Hits

During the operation of the **DoCollision()** routines, if you have enabled boundary collisions for a GEL and that GEL crosses a boundary, the system calls the boundary-hit routine you have defined. Note that the system calls the routine once for each GEL that has gone outside of the boundary.

The system will call your routine with the following two arguments:

- o A pointer to the **VSprite** structure of the GEL that hit the boundary
- o A flag word containing one to four bits set, representing top, bottom, left and right boundaries, telling you which one or more boundaries it has hit or exceeded. To test these, use the names **TOPHIT**, **BOTTOMHIT**, **LEFTHIT**, and **RIGHTHIT**.

GEL-to-GEL Collisions

If, instead of a GEL-to-boundary collision, **DoCollision()** senses a GEL-to-GEL collision, the system calls your collision routine with the following two parameters. They will be different from those in the GEL-to-boundary collision.

- o Address of the **VSprite** structure that defines the uppermost (or leftmost if y coordinates are identical) object of a colliding pair
- o Address of the **VSprite** structure that defines the lowermost (or rightmost if y coordinates are identical) object of a colliding pair

Handling Multiple Collisions

When multiple elements collide within the same display field, the following set of sequential calls to the collision routines occurs:

- o The system issues each call in a sorted order for GELs starting at the upper left-hand corner of the screen and proceeding to the right and down the screen.
- o For any two colliding graphics elements, the system issues only one call to the collision routine for this pair. The system bases the collision call on the object that is the highest and leftmost of the pair on the screen.

Preparing for Collision Detection

Before you can use the system to detect collisions between GELS, you must initialize the table of collision-detection routines. This table points to the actual routines that you will use for the various collision types you have defined. Also, you must prepare certain variables and pointers within the **VSprite** structure: **BorderLine**, **CollMask**, **HitMask**, and **MeMask**.

Building a Table of Collision Routines

To add to or change the table entries for the collision routines, call the **SetCollision()** routine. The syntax for this routine follows:

SetCollision(num, routine, Ginfo)

where

num

is the collision vector number

routine

is a pointer to the user collision routine

GInfo

is a pointer to a **GelsInfo** structure

When the **View** structure is first initialized, the system sets all of the values of the collision routine pointers to zero. You must initialize those table entries so that they correspond to the **HitMask** and **MeMask** bits that you have set. Only those table entries can cause the system to call the collision routines.

You must also allocate a table, pointed to by **GelsInfo**, for vectors. The table needs to be only as large as the number of bits for which you wish to provide collision processing. For example:

```

VOID myCollisionRoutine( GELM, GELN ) /* sample collision routine */
struct VSprite *GELM;
struct VSprite *GELN;
{
    printf("GEL at %lx has hit GEL at %lx", (long)GELM, (long)GELN);
}

/* sample initialization */
ReadyGels(gelsinfo, rastport); /* use exec_support function */
SetCollision( 15, myCollisionRoutine, &gelsinfo );

```

Collision Mask

The variable named **CollMask** is a pointer to a memory area that you have reserved for holding the collision mask of a GEL. A collision mask is usually the same as the shadow mask of the GEL, formed from a *logical-or* combination of all 1 bits in all planes of the image. Figure 3-3 shows an example collision mask.

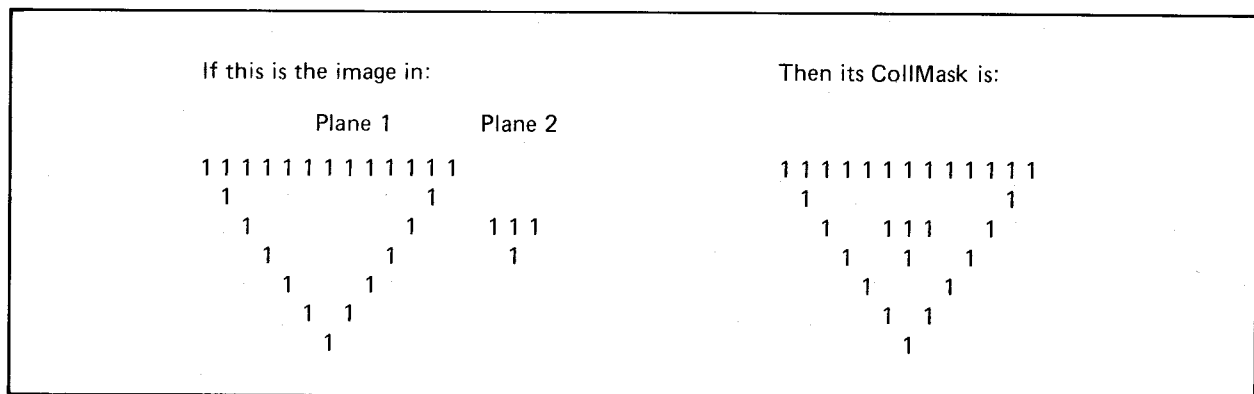


Figure 3-3: A Collision Mask

You normally use this collision mask to control drawing of the object and to define essentially the positions where there is an image bit present. After you have defined the collision mask through the routine **InitMasks()**, you may specify that the system is to store both the shadow mask and the collision mask in the same location.

For example, here are typical program statements to reserve an area for the sprite shadow, initialize the pointer correctly, and then specify that the system uses the same mask for collisions (this example assumes a two-word-wide, four-line-high image):

```

/* reserve 8 16-bit locations for sprite
 * shadow to be stored into by the system.
 */
WORD myShadowData[8];
    /* and point to it */
myVSprite.ImageShadow = myShadowData;
    /* collision mask is same as shadow */
myVSprite.CollMask = myShadowData;

```

As an alternative, for certain game-oriented applications, you may design certain objects with sensitive regions and non-sensitive regions. Suppose you have an object, such as a spaceship, with an outer layer that is to be non-sensitive and an inner core that is to register collisions for the overall object. You would define your shadow mask with 1 bits in the appropriate positions to define the desired sensitive area. An example using this type of image is shown in figure 3-4.

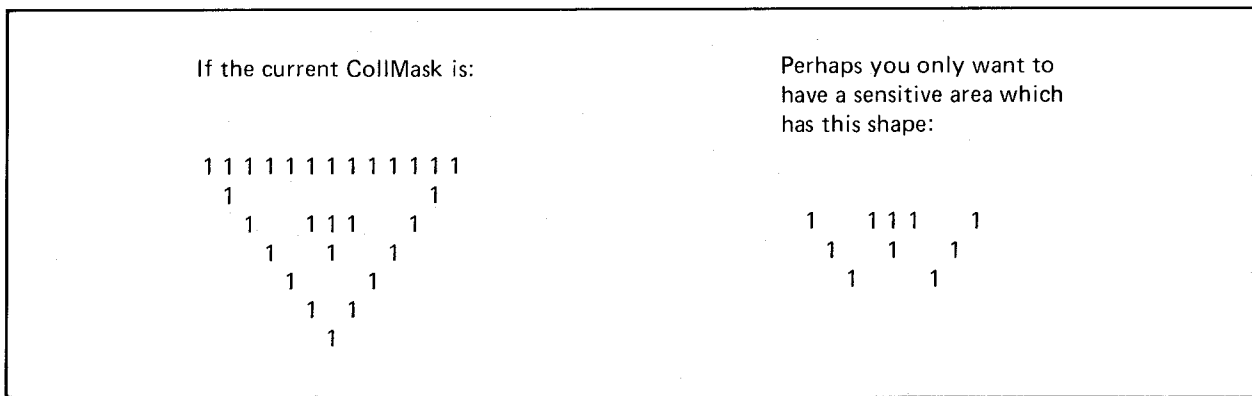


Figure 3-4: Shadow Mask for a Sensitive Area

BorderLine Image

For fast collision detection, the system uses the pointer named **BorderLine**. **BorderLine** specifies the location of the horizontal *logical-or* combination of all of the bits of the object. It may be compared to taking the whole object and squashing it down into one single horizontal line. Here is a sample of an object and its **BorderLine** image:

OBJECT

```
001100001100
000110011000
000011110000
000110011000
001100001100
```

BORDERLINE IMAGE

```
00111111100
```

The borderline image establishes a single set of words (represented by the collision mask) that have 1 bits at the outermost edges of the object. Using this squashed image, the system can quickly determine if the image is touching the left or rightmost boundary of the drawing area.

To establish the borderline data, you make a system call to **InitMasks()**. Before calling **InitMasks()**, you provide the system with a place to store the image it creates. The size of the data area you reserve must be at least as large as the image is wide.

In other words, if it takes three 16-bit words to hold the width of a GEL, then you must reserve three words for the borderline image. For example:

```
/* reserve some space for the border image to be stored for this Bob */
WORD myBorderLineData[3];

/* tell the system where to put the BorderLine image it will form */
myVSprite.BorderLine = myBorderLineData;
```

Note: Both **Bobs** and **VSprites** participate in the software collision detection.

The next section tells how to turn on the software collision detection independently for each GEL.

Software Collision-Detect Control Variables

You can enable or disable software collision detection for each GEL independently. In addition, any time the system senses a collision, you can specify which of 16 possible collision routines you wish to have automatically executed. The **HitMask** and **MeMask** variables in the **VSprite** structure let you specify the relationships between different GELS.

By specifying the bits in these masks, you can control how and when the system senses collisions between objects. The collision testing routine, in addition to sensing an overlap between objects, also uses these masks to determine which routine(s) (if any) the system will call when a collision occurs.

When the system determines a collision, it *ands* the **HitMask** of the upper-leftmost object in the colliding pair with the **MeMask** of the lower-rightmost object of the pair. The bits that are 1s after the *and* operation choose which of the 16 possible collision routines to perform.

- o If the collision is with the boundary, bit 0 is a 1 and the system calls the collision handling routine number 0. You assign bit 0 to the condition called "boundary hit." The system uses the flag called BORDERHIT to indicate that an object has landed on or moved beyond the outermost bounds of the drawing area (the edge of the clipping region).
- o If you set any one of the other bits (1 to 15), then the system calls the collision handling routine corresponding to the set bit.

If more than one bit is set in both masks, the system calls the vector corresponding to the rightmost bit.

Using HitMask and MeMask

This section provides an example of the use of the **HitMask** and **MeMask** to define a new form of collision detection.

Suppose there are two classes of objects that you wish to control on the screen: ENEMYTANK and MYMISSILE. Objects of class ENEMYTANK should be able to pass across one another without registering any collisions. Objects of class MYMISSILE should also be able to pass across one another without collisions. However, when MYMISSILE collides with ENEMYTANK or ENEMYTANK collides with MYMISSILE, the system should process a collision routine.

Choose a pair of collision detect bits not yet assigned within **MeMask**, one to represent ENEMYTANK, the other to represent MYMISSILE. You will use the same two bits in the corresponding **HitMask**.

	<u>MeMask</u>	<u>HitMask</u>	
Bit #	2 1	2 1	
GEL #1	0 1	1 0	ENEMYTANK
GEL #2	0 1	1 0	ENEMYTANK
GEL #3	1 0	0 1	MYMISSILE

In the example, bit 1 represents ENEMYTANK objects. In the **MeMask**, bit 1 is a 1 for GEL #1 and says "I am an ENEMYTANK." Bit 2 is a zero says this object is *not* a MYMISSILE object.

In bit 1 of the **HitMask** of GEL #1, the 0 bit there says, "I will not register collisions with other ENEMYTANK objects." However, the 1 bit in bit 2 says, "I *will* register collisions with MYMISSILE objects."

Thus when a call to **DoCollision()** occurs, for any objects that appear to be colliding, the system *ands* the **MeMask** of one object with the **HitMask** of the other object. If there are non-zero bits present, the system will call one (or more) of your collision routines.

In this example, suppose that the system senses a collision between ENEMYTANK #1 and ENEMYTANK #2. Suppose also that ENEMYTANK #1 is the top/leftmost object of the pair. Here is the way that the collision testing routine performs the test to see if the system will call any collision-handling routines:

<u>Bit #</u>	<u>2</u>	<u>1</u>
ENEMYTANK #1 MeMask	0	1
ENEMYTANK #2 HitMask	1	0
Result of <i>and</i>	0	0

Therefore, the system does not call a collision routine.

Suppose that **DoCollision()** finds an overlap between ENEMYTANK #1 and MYMISSILE, and MYMISSILE is the top/leftmost of the pair:

<u>Bit #</u>	<u>2</u>	<u>1</u>
MYMISSILE #1 MeMask	1	0
ENEMYTANK #2 HitMask	1	0
Result of <i>and</i>	1	0

Therefore, the system calls the collision routine at position 2 in the table of collision-handling routines.

BOB/VSPRITE COLLISION BOUNDARIES WITHIN A RASTPORT

To specify a region within the **RastPort** (drawing area) that the system will use to define the outermost limits of the GEL boundaries, you use the following variables: **topmost**, **bottommost**, **leftmost**, and **rightmost**. The **DoCollision()** routine tests these boundaries when determining collisions within this **RastPort**.

Here is a typical program segment that assigns the variables correctly. It assumes that you already have a **RastPort** structure named **myRastPort**.

```
myRastPort->GelsInfo->topmost = 50;
myRastPort->GelsInfo->bottommost = 100;
myRastPort->GelsInfo->leftmost = 80;
myRastPort->GelsInfo->rightmost = 240;
```

The current release of the system software makes use of the clipping-rectangle feature of the **RastPorts** to create clipping to the **RastPort**'s limits. However, you may base the "boundary collision" limits for this **RastPort** on the variables shown here.

ADDING NEW FEATURES TO BOB/VSPRITE DATA STRUCTURES

This section describes how to expand the size and scope of the **VSprite** or **Bob** data structures. In the definition for the **VSprite** and the **Bob** structures, there is an item called **UserExt** at the end of the structure. If you want to add something to these structures (specifically, a user extension), you simply specify that the **UserExt** variable is composed of a specific type.

Why would you want to add things to the structure? When the **DoCollision()** routine passes control to your collision-processing function, you may wish to change some variable associated with the GEL. The example below places speed and acceleration figures with each GEL. When

you perform the collision routine, it exchanges these values between the two colliding objects. The system uses additional routines during the no-collision times to calculate the new positions for the objects.

You could define a structure similar to the following:

```
struct myInfo {
    short xvelocity;
    short yvelocity;
    short xaccel;
    short yaccel;
};
```

that you want to have associated with each of the GELS. These variables are, for example, *your* user extensions.

You would also provide the following line:

```
For VSprites:
    #define VUserStuff struct myInfo
```

```
For Bobs:
    #define BUserStuff struct myInfo
```

```
For AnimObs:
    #define AUserStuff struct myInfo
```

When the system is compiling the *graphics/gels.h* file with your program, the compiler substitutes “struct myInfo” everywhere that **UserExt** is used in the header. The structure is thereby customized to include the items you wish to associate with it.

Note: The header files include the following UserStuff variables for **VSprites**, **Bobs**, and **AnimObs**:

```
VSprites:    VUserStuff
Bobs:       BUserStuff
AnimObs:   AUserStuff
```

Animation Structures and Controls

This section outlines the system animation support for **Bobs** only. In the section called “Bob Priorities” you learned how to control the priorities of **Bobs** with respect to one another by specifying the drawing sequence. The following sections explain how to link objects and how to specify an animation completely by linking different views of objects into a sequence.

To perform animation, an artist produces a series of drawings. Each drawing differs from the preceding one so that when they are sequenced, the object appears to move naturally. An animation in the Amiga consists of a linked list of the components of the animation object and each component as a linked list of the different drawings in its sequence.

To perform the actual animation, you make a call to a system routine called **Animate()**. When you call **Animate()**, the software follows all of your animation instructions and “moves” the objects accordingly. When you next call **DrawGList()**, the system draws all objects in the position caused by your calls to **Animate()**. Essentially, **Animate()** simply manipulates a set of instructions in a set of object lists. Only when the system draws the objects are your instructions displayed visually.

Remember, the system draws the currently sorted objects from its GELS list.

CHARACTERISTICS OF THE ANIMATION SYSTEM

The animation system lets you define a series of **Bobs**, which it then links into an overall object. The combined object consists of one or more **Bobs** that comprise the overall object and additional **Bobs** that comprise alternate appearances (animation sequences) for the various component parts.

You specify the following:

- o The initial appearance of an overall object by defining **Bobs** as its components
- o Alternate views of various components by defining additional **Bobs**
- o The drawing precedence for the initial appearance of the object among the **Bobs** that comprise the initial appearance

The animation system does the following:

- o Moves all linked objects simultaneously

- o Maintains inter-object prioritization
- o Sequences alternate views to provide animation through user-specified timing variables

KEEPING TRACK OF GRAPHIC OBJECTS

The section called “Getting the List of Bobs in Order” described how the system maintains a list of **Bobs** to draw on the screen according to your instructions. The animation system selectively adds items to and removes items from this list of screen objects during the **Animate()** routine. The next time you call **DrawGList()**, the system will draw the current **Bobs** in the list into the selected **RastPort**.

CLASSES OF ANIMATION OBJECTS

You have two classes of animation objects to consider: **AnimObs** and **AnimComps**. The **AnimOb** is the primary animation object. It is this object whose position you are specifying with respect to the coordinates of the drawing area. Actually, an **AnimOb** itself contains no imagery. It is merely the top-level data structure that organizes the components that it manages and that specifies a position relative to which everything else is drawn. The **AnimComp**, on the other hand, is an animation component — for example, an arm, leg, or head — of an animation object. The animation object consist of animation components that you specify.

To define an **AnimOb**, you specify several characteristics of the primary animation object, including the following:

- o The initial position of this object
- o Its velocity and acceleration in the X and Y directions
- o How many calls to **DrawGList()** you have made while this object has been active
- o A pointer to a special animation routine related to this object (if desired)
- o A pointer to the first of its animation components
- o Your own extensions to this structure, if desired

POSITIONS OF ANIMATION OBJECTS

The next two sections tell how to specify the initial position of an **AnimOb** and its **AnimComp**.

Position of an AnimOb

To specify a registration point within the drawing area (the **RastPort**) for all components, you use the variables **AnX** and **AnY** in the **AnimOb** structure. Figure 3-5 illustrates that each component has its own offset from the object's registration point.

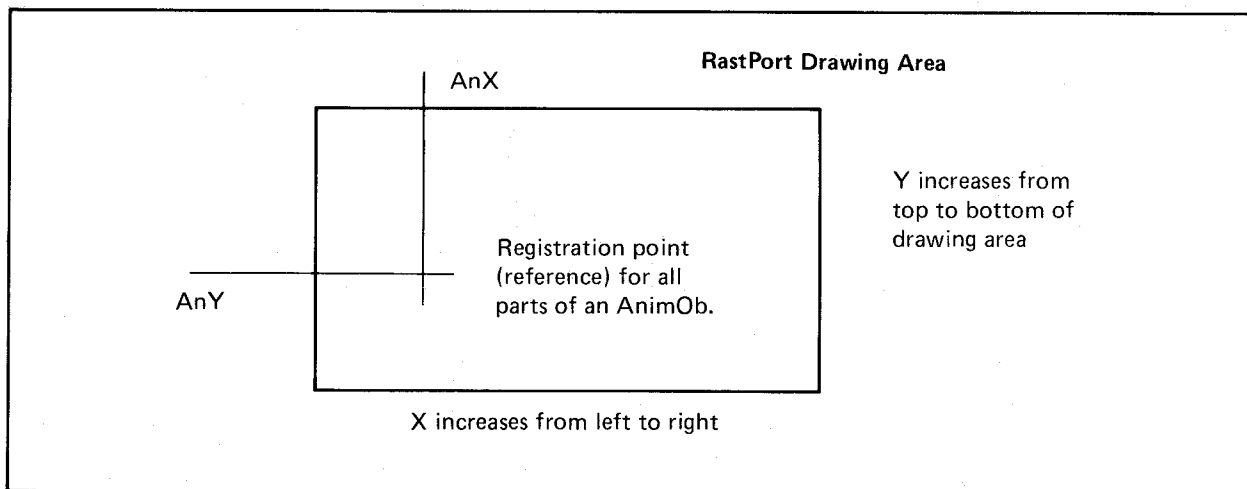


Figure 3-5: Specifying an AnimOb Position

Position of an AnimComp

To specify where the component is to be located *relative to the position of the registration point*, you use variables in the **AnimComp** structure. When you move the animation object, all of the component parts of this animation object move with it, as illustrated in figure 3-6.

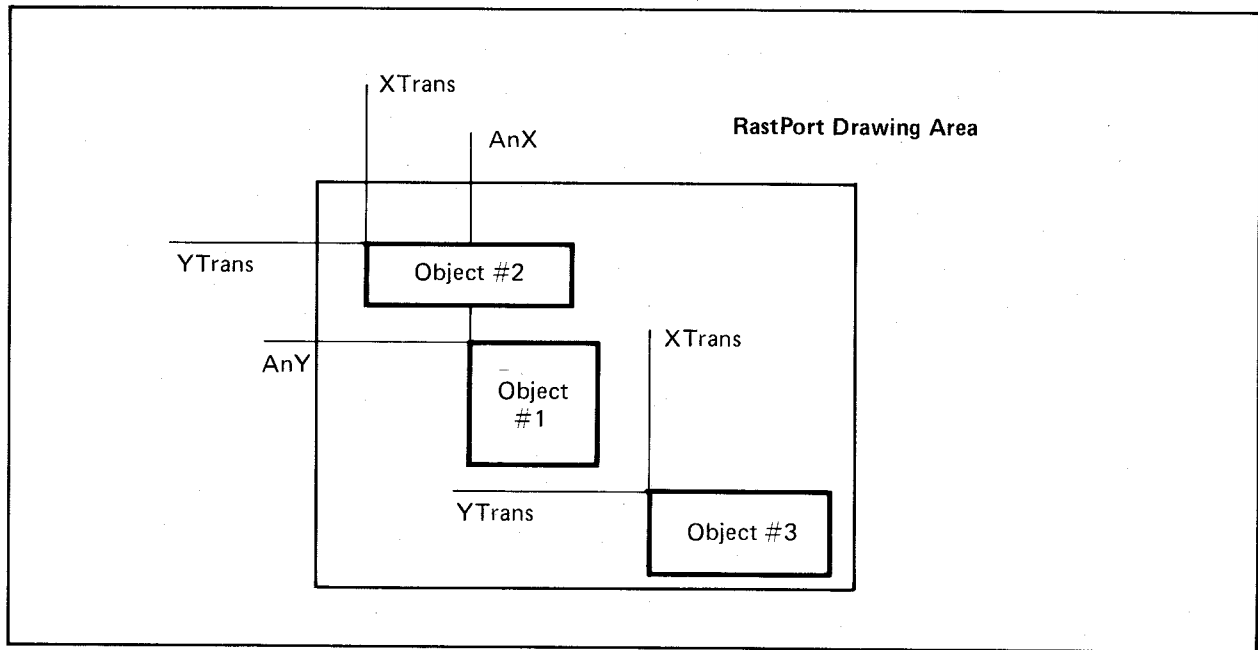


Figure 3-6: Specifying an AnimComp Position

To specify the relative placement of a component with respect to the registration point of the **AnimOb**, you assign the values of **XTrans** and **YTrans** in the **AnimComp** structure. These values can be positive (as shown for object #3), negative (as shown for object #2), or zero (as shown for component #1) in figure 3-6 above.

Now that the system knows the position of the objects and components you wish to animate, you can tell the system how to animate them. The following sections describe the animation choices provided for you by the system.

ANIMATION TYPES

The system software allows two forms of animation: sequenced drawing and motion control.

Sequenced Drawing

In sequenced drawing, an artist produces a sequence of views of an object, where each view is a modification of a preceding view. To produce apparent motion of the object, the artist draws each new view of an object at a position somewhat farther from a common reference point than the preceding view.

If an animation is to be continuous, based on a repeating sequence, then the last drawing in the series should allow the first drawing in the series to be the next in line, creating a continuity of motion. Figure 3-7 shows four out of a sequence of drawings that could use this technique for animation. (The other intermediate steps are not shown.)

As you will notice, each of the drawings, reading from right to left, is a little closer to its registration point (the reference point). The upper level of the figure shows the figures individually. The lower level shows the figures overlaid, demonstrating that smooth motion would be possible. To the left of the overlaid figures is a second set, drawn in gray, representing the reinitialization of the sequence of drawings, beginning with number one.

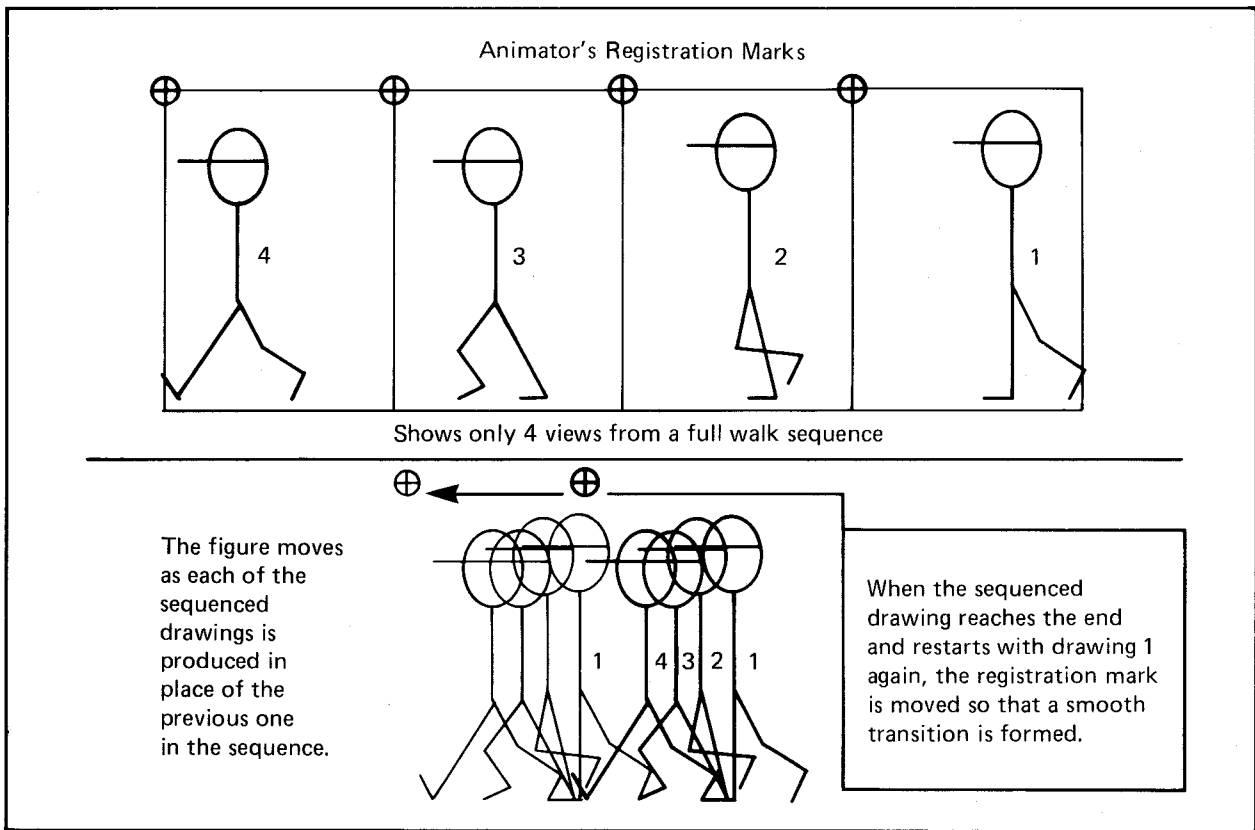


Figure 3-7: A Sequenced Drawing

Sequenced animation often consists of a closed “ring” of drawings. When the last drawing of the sequence has been completed, the first drawing in the sequence is repeated again, becoming the first in the next part of the animation, offset by a specific position in space.

To specify sequenced drawing, use the variables called **compFlags** in the **AnimComp** structure, and **RingXTrans** and **RingYTrans** in the **AnimOb** structure.

To move the registration mark to a new location, you set the RINGTRIGGER bit for a component in its **compFlags** variable. The system software adds the values of **RingXTrans** and **RingYTrans** found in the **AnimOb** structure to the values of **AnX** and **AnY** of the head object (the registration mark), thereby moving the reference point to the new location. The next time you execute **DrawGList()**, the drawing sequence starts over again at the new location, mating properly with the final drawing of the sequence at the old registration mark.

You usually set RINGTRIGGER in only one of the animation components in a sequence; however, you can choose to use this flag and the translation variables in any way you wish.

Motion Control

In the second form of animation, you can specify objects that have independently controllable velocities and accelerations in the X and Y directions. Components can still sequence. Furthermore, you can use ring and velocity simultaneously if you wish.

The variables that control this motion are located in the **AnimOb** structure and are called:

- o **YVel, XVel**—the velocities in the y and x directions
- o **YAccel, XAccel**—the accelerations in the y and x directions

Velocities and accelerations can be either positive or negative.

The system treats the velocity numbers as though they are fixed-point binary fractions, with the decimal point fixed at position 6 in the word. That is:

vvvvvvvvvv.ffffff

where v stands for actual values that you add to the x or y (**AnX, AnY**) positions of the object for each call to **Animate()**, and f stands for the fractional part. By using a fractional part, you can specify the speed of an object in increments as precise as 1/64th of an interval.

In other words, if you set the value of **XVel** at 0x0001, it will take 64 calls to the **Animate()** routine before the system will modify the object's x coordinate position by a step of one. The system requires a value of 0x0040 to move the object one step per call to **Animate()**.

Each call you make to **Animate()** simply adds the value of **XAccel** to the current value of **XVel**, and **YAccel** to the current value of **YVel**, modifying these values accordingly.

Using Sequenced Drawing and Motion Control

If you are using sequenced drawing, you will probably set the velocity and acceleration variables to zero. This allows you to produce the animation exactly in the form in which the artist has designed it in the first place.

Consider an example of a person walking. As each foot falls, with sequenced drawing, it is positioned on the ground exactly as originally drawn. If you include a velocity value, then the person's foot will not be stationary with respect to the ground, and the person appears to "skate" rather than walk. If you set the velocity and acceleration variables at zero, you can avoid this problem.

INITIALIZING THE ANIMATION SYSTEM

To initialize the system, you must define a pointer to an **AnimOb**. The system uses this pointer to keep track of all of the real **AnimObs** that you create. The following typical code sequence accomplishes this:

```
struct AnimOb *animKey;
-
-
animKey = NULL;
```

Note: Before you can use the animation system, you must call the routine **InitGels()**. Therefore, you must initialize the GEL system as well as the animation system. See the "Initializing the GEL System" section for details on **InitGels()**, the Bob-control system that eventually displays the objects that you manipulate.

SPECIFYING THE ANIMATION OBJECTS

To add animation objects to the controlled object list, you use the routine **AddAnimOb()**. Figure 3-8 shows how to build a list of controlled objects using this routine. The **animKey** always points to the object most recently added to the list.

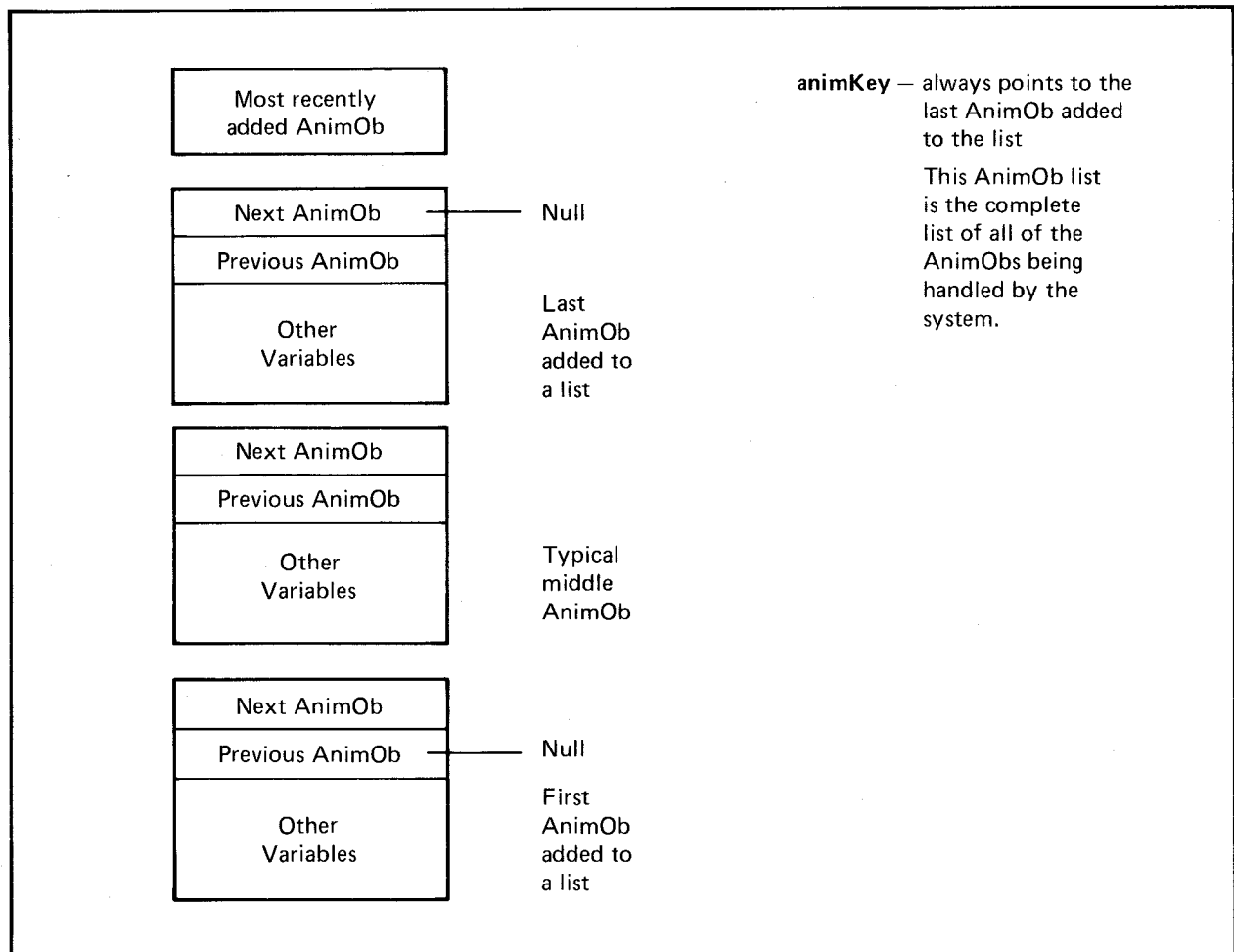


Figure 3-8: Linking AnimObs into a List

Next, you tell the system about the components that make up the object.

SPECIFYING ANIMATION COMPONENTS

As previously stated, each animation object consists of one or more individual component parts. The parts may be, for example, the body of an object, its arms, its legs, and so on. Not only does the system animator move parts from place to place, but it also offers different views of each of the parts. To specify the relationships between the individual parts and views of those parts, you initialize various pointers within the **AnimComp** structure.

You use the pointers called **PrevSeq** and **NextSeq** to build a doubly-linked list of a set of animation components used for sequenced drawing, as outlined above. In all cases, when you specify **AnimComps**, you must initialize these pointers to build the sequence that you wish the

system to follow for drawing the various views of this component. The “Animation Sequencing” section below shows how the system uses these pointers.

To link the components together into a whole object, use the pointers called **PrevComp** and **NextComp**. When you build an animation object, you must initialize the **PrevComp** and **NextComp** pointers *for only the initial view of the animation object*. Whenever the animation system senses that one of the animation objects has “timed out” and switched to a new sequence of that component, the system automatically adjusts the **PrevComp** and **NextComp** pointers so that it retains the complete animation object.

Figure 3-9 shows an animation object built of several components. The **AnimOb** points to the head component. Notice that the “head” component may be any one of the components of the object. A pointer in the structure of the head component, in turn, points to the next one, and so on (building the initial view of the object).

To point around the ring for each of the component sequenced views (although the objects do not necessarily have to form a ring), you initialize the sequence pointers **NextSeq** and **PrevSeq**. The animation system ignores the **PrevComp** and **NextComp** pointers for each of the *non-current* components.

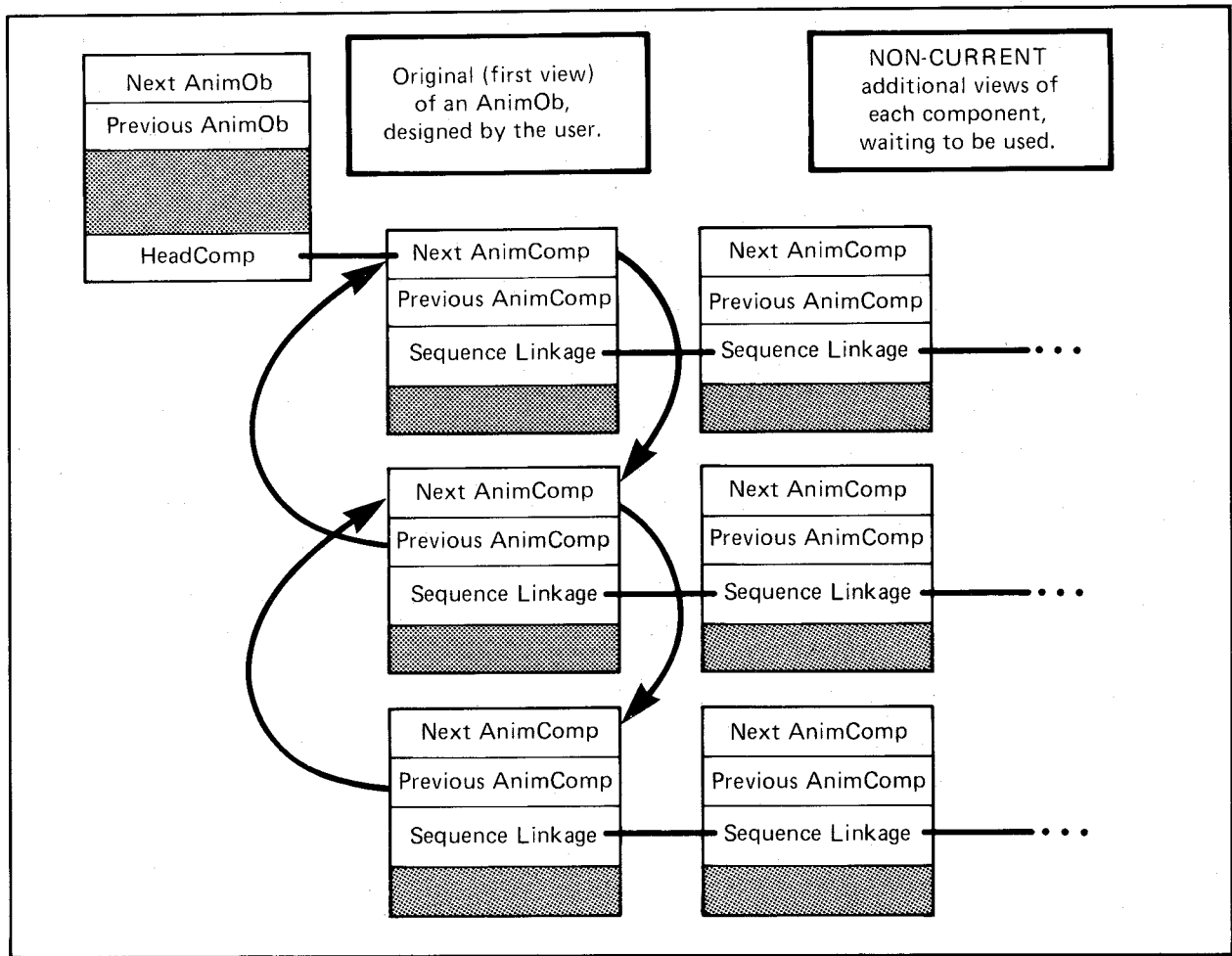


Figure 3-9: Linking AnimComps To Form an AnimOb

DRAWING PRECEDENCE

The sequence in which you link the components in a list to define the object itself is immaterial. The system simply uses this list of components to define the overall object. To specify the drawing precedence for the objects in an animation object, you use the **Before** and **After** pointers in the **Bob** structure *for the initial sequence of the animation object*.

If you refer to the description of adding **Bobs** in the section called "Adding a Bob," you will see that when you add **Bobs** to the system, the **Before** and **After** pointers control the drawing sequence and thereby the precedence of the objects. Once you have added the **Bobs** to the system with **AddBob()**, you must assign a fixed set of pointers to establish the correct drawing order.

Animation components may have several views, each of which points to a **Bob** structure. However, only one of those views is actually “active” for that component at any one time, making up part of the overall animation object. The animation system adjusts the **Before** and **After** pointers of the **Bob** structure for each of the current views to maintain the sequence of drawing for each of the components the same as that you have defined for the initial view. Adjustments take place in the sequencing any time any one of the animation components “times out” and shows a new sequence. Therefore, if you are defining **Bobs** as part of the animation system, you need only initialize the **Before** and **After** pointers within the **Bob** structure for the initial sequence of each of the components.

You may wish to define multiple animation objects. To assure that one complete *object* always has priority over another object, you can use the initial sequence linkage to control this as well. You use the **Bob Before** and **After** pointers to link together the last **AnimComp**’s **Bob** of one **AnimOb** to the first **AnimComp**’s **Bob** of the next **AnimOb**. The system maintains the drawing order during calls to **Animate()** from that time onward.

You may modify the drawing order during part of the animation (such as to make one object pass in front of another during one display sequence, then pass behind it on the next sequence). You can perform this kind of activity, if you wish, during an **AnimORoutine** or **AnimCRoutine**. See the section called “Your Own Animation Routine Calls” for details.

ANIMATION SEQUENCING

To perform sequenced drawing, you must define the sequence in which you wish the drawings to be made. For each of the animation components, there is a set of pointers that allows you to define the exact sequence in which the drawings should appear.

After a period of time that you have specified, which is separately controllable for each component, the system software automatically switches from the current drawing in the sequence to the next one. For this purpose, you provide three pieces of information in the **AnimComp** structure: pointers to the previous and next drawings in the sequence that you have defined, a user flag variable called **Flags**, and a **TimeSet** variable.

After the specified time interval for each of the sequenced drawings, the system software switches to show the next drawing specified in the sequence. The next section shows how you specify the time.

Figure 3-10 illustrates how the system uses the “next sequential image” pointer to step from one image to the next at the specified time.

If you set the **RINGTRIGGER** bit in the **Flags** variable, the system adjusts the reference point for the sequenced drawing. See the “Sequenced Drawing” section above for details.

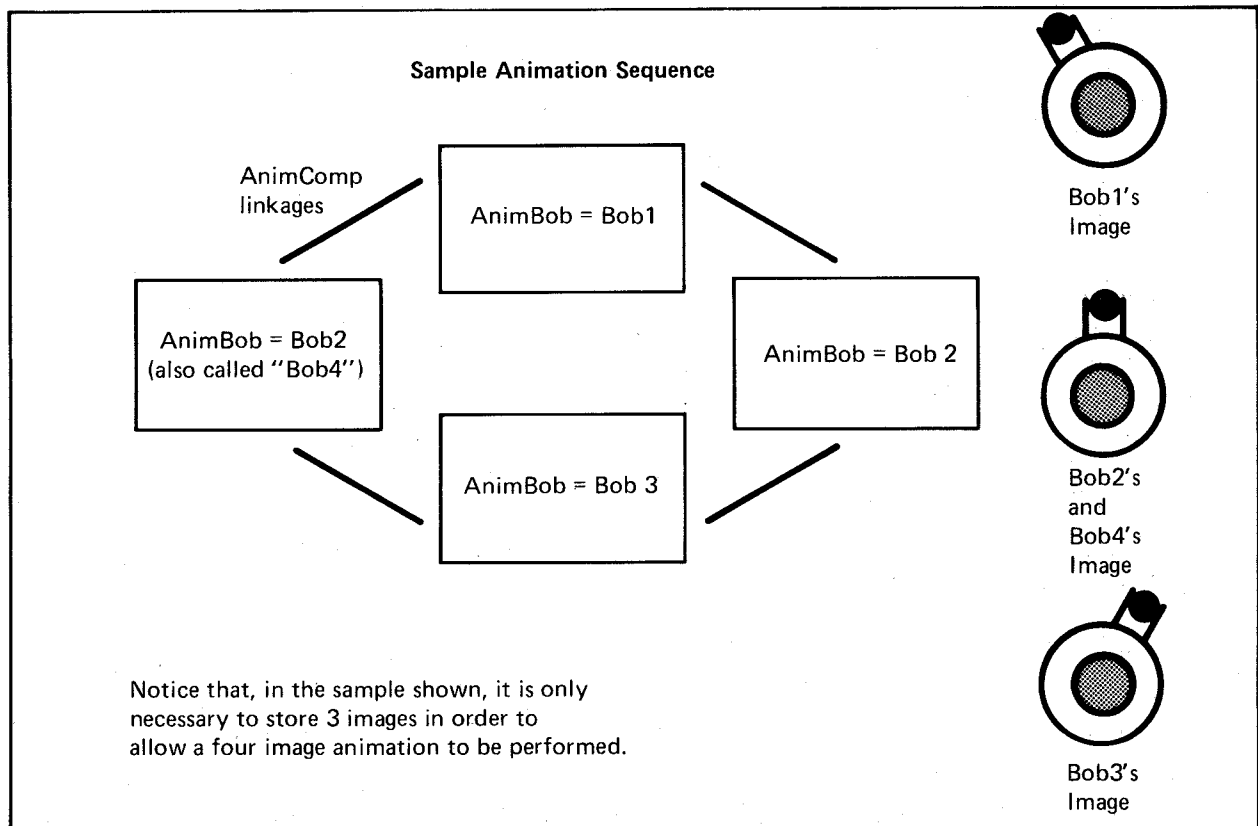


Figure 3-10: Linking AnimComps for Sequenced Drawing

SPECIFYING TIME FOR EACH IMAGE

When you have defined all of your animation objects and components, you call the **Animate()** routine. To manipulate the objects, you set the variable called **Timer** in the **AnimComp** structure and you set a corresponding variable called **TimeSet** (also in the **AnimComp** structure).

When the system selects the animation component, the system copies the value currently in **TimeSet** into the variable named **Timer**. If **Timer** has a nonzero value when you call **Animate()**, then the current view of the animation component remains the active view for as many calls to **Animate()** as you specify with the value in **Timer**. When the **Timer** value counts down to zero, the system makes the next sequential view active. If you set the value in **TimeSet** to zero, **Timer** remains zero. **Timer** never triggers from a non-zero state and, therefore, does *not* cause any change in the view.

When the system activates a new sequence component, it checks that component's **compFlags** to see if the RINGTRIGGER flag bit is set. If so, the system performs ring processing, which means that it adds the values **RingYTrans** and **RingXTrans** to **AnY** and **AnX** respectively. See the section called "Animation Types" for details.

Now let's see how this process works in an actual animation. Let's say that you are animating the figure of a man. As he walks across the screen, he swings his arm back and forth at a fixed rate. Assume that you have three drawings of the arm: swung forward, at a center position, and swung back. To animate the arm, you may follow these steps:

1. Define four **Bobs**: the first for the forward swing, the second for the center, the third for the back swing, and the fourth centered again.
2. Define four **AnimComps**, one for each of these **Bobs**. To link them together in a sequence (forward, center, back, center), use the **PrevSeq** and **NextSeq** pointers.
3. Link *one* of the **AnimComps** in this sequence to the **AnimComp** that defines the body of the man, using the **AnimComp**, **PrevComp**, and **NextComp** pointers.
4. Set the **Timer** variable for *each* sequenced **AnimComp** to a value appropriate for him to hold that pose. For example, three calls to **Animate()** for forward and back, and two calls for each of the two centered positions of his arm might be appropriate values.
5. Set the value of **XTrans** and **YTrans** for *each* **AnimComp** to position the arm properly with respect to the rest of the body for each sequence of the arm swing.
6. Continue the arm sequence by setting the RINGTRIGGER bit in the flags variable of the last sequence, thereby triggering a return to the first view when the timer of the last view times out.

Now, each time you call **Animate()**, the animation system checks all of the **Timer** variables, as well as calling your **AnimCRoutines** and **AnimORoutines**. When each of the **Timer** variables becomes a zero, the next sequenced view of the **AnimComp** replaces the current sequence. When an **AnimComp** becomes "current," the value in its **TimeSet** variable is copied into its **Timer** variable.

This also means that you have told the system two things: first, to remove the **Bob** of the current sequence from the system **Bob** list the next time you call **DrawGList()**; and second, to use the **Bob** representing the new sequence in its place. The system automatically copies the **Bob Before** and **Bob After** pointers from the current sequence into the new sequence **AnimComp's Bob** to assure that the object is still drawn in the same order, maintaining its priority relative to other objects in the drawing area.

YOUR OWN ANIMATION ROUTINE CALLS

The **AnimOb** and **AnimComp** structures include pointers to your own routines that you want the system to call. If you want a routine to be called, you must specify the address of the routine in this variable. If no routine is to be called, you must set this variable to zero. No values are passed to these routines, except a pointer to its **AnimOb** or **AnimComp**, respectively. However, because you set each AnimORoutine (the **AnimOb** routine) and AnimCRoutine (the **AnimComp** routine), you can use the extensions to the **AnimOb** or **Bob** or **VSprite** structures to hold the variables you need for your own routines.

Suppose you are creating the following animation:

- A man is walking a dog down a street. There is a fireplug at one side of the screen. Let's say you wish to change the appearance of the fireplug if the dog approaches too closely. You would, therefore, design an AnimORoutine to do a proximity check on the dog.
- To allow the fireplug to have different appearances, you might provide three individual views. One is normal, one is an intermediate view (comparable to the center arm-swing mentioned earlier), and the final view is a "strength pose," saying "back off dog!"
- You may set the **TimeSet** and **Timer** variables for the "normal" appearance for the fireplug at zero. This means that it should never change from this appearance no matter how many calls to **Animate()** occur, as defined above. (If it is already zero, it will not decrement; therefore, it can never go from non-zero to zero).
- You may set the **TimeSet** variable for the intermediate view to 1 (stay in the intermediate pose for only one call to **Animate()**). In addition, you may set the **TimeSet** variable for the strength pose to 10 (stay strong for ten calls to **Animate()**).
- For each call to **Animate()**, the AnimORoutine for the fireplug checks how close the dog has approached. If it is within a certain range, the AnimORoutine changes the **Timer** variable for the normal fireplug pose to a 1.
- The next call to **Animate()** finds a value of 1 in the **Timer** variable and decrements it. This makes a value of 0, forcing a change to the next sequence (the intermediate pose). The system will remove the normal pose **Bobs** from the system **Bob** list it is to draw, and the next call to **DrawGLList()** will therefore draw the intermediate pose instead.
- The next call to **Animate()** finds a value of 1 in the **Timer** variable for the intermediate pose and decrements it, causing a change to the strength pose. The fireplug remains in the strength pose for ten calls to **Animate()**, returning through the intermediate pose for one call, then to the normal pose again.

- o Now that the **Timer** value has become zero again, the fireplug returns to the original state, staying in its normal pose until the dog again approaches within range.

MOVING THE OBJECTS

When you have defined all of the structures and have established all of the links, you can call the **Animate()** routine to move the objects. **Animate()** adjusts the positions of the objects as described above, and calls the various subroutines (**AnimCRoutines** and **AnimORoutines**) that you have specified.

After the system has completed the **Animate()** routine, as the screen objects have been moved, their order in the graphics objects list may possibly be incorrect. Therefore, as always, before ordering the system to redraw the objects, you must sort them first.

If you perform **DoCollision()** when the system has newly positioned the objects after your call to **Animate()**, your collision routines may also have an effect on the ultimate position of the objects. Therefore, you should again call **SortGList()** to assure that the system correctly orders the objects before you call **DrawGList()**, as illustrated in the following typical call sequence:

```
/* ... setup of graphics elements and objects */

Animate( key, rp );      /* "move" objects per instructions */
SortGList( rp );        /* put them in order */
DoCollision( rp );      /* software collision detect/action */
SortGList( rp );        /* put them back into right order */
DrawGList( vp, rp );    /* draw into current RastPort */
```

Complete Example Program

The following program produces a single-buffered display with two **Bobs** and two **Vsprites**.

```

/* SAMPLE PROGRAM THAT USES GELTOOLS TO PRODUCE A DOUBLE BUFFERED DISPLAY
 * SCREEN CONTAINING TWO BOBS AND TWO VSPRITES
 *
 * Author: David Lucas
 */

/* Leave this structure definition at the top. Look at gels.h. */
struct vInfo {
    short vx,vy;          /* This VSprite's velocity. */
    short id;
};
#define VUserStuff struct vInfo

/* Things to notice:

Default value in sprite/playfield priority register has all
hardware sprites having a higher priority than either of the
two playfields. Areas containing color 0 of both the bob and
vsprite are shown as transparent (see hole in center of each).

You can specify bob drawing order by using the before and after
pointers, thereby always maintaining an apparent precedence of
one bob over another. Re Vsprites... because they are assigned
sequentially from top of screen to bottom, in sprite numerical
order (0, 1, 2, 3 etc), and because the lowest numbered hardware
sprite has the highest video precedence, the sprite that is
closest to the top of the screen always appears in front of the
sprite beneath it.

Without double-buffering, there would be flicker on the part
of the bobs. Double buffering consists of writing into an area
that is not being displayed. Some of the flicker could have been
alleviated by waiting for the video beam to reach top-of-frame
before doing the drawing, but when the bobs are near the top,
it makes it all the more difficult to draw without apparent
flicker in that case. Also note that multitasking will
occasionally upset even this plan in that it can delay the
drawing operation until the beam is in the area that is being drawn.

*/

/*****
 * A sprite and a bob on a screen.
 */

#include "intuall.h"

#define SEMWIDTH 320      /* My screen size constants. */
#define SEMHEIGHT 200
#define SEMDEPTH 4

#define REMWIDTH 330     /* My rastport size constants. */
#define REMHEIGHT 210
#define REMDEPTH SEMDEPTH

#define VSPRITEWIDTH 1   /* My VSprite constants. */
#define VSPRITEHEIGHT 12
#define VSPRITEDEPTH 2
#define NSPRITES 2

#define BOBWIDTH 62     /* My Bob constants. */
#define BOBHEIGHT 31
#define BOBDEPTH 4
#define NBOBS 2

struct IntuitionBase *IntuitionBase = NULL;
struct GfxBase *GfxBase = NULL;

struct IntuiMessage *MyIntuiMessage = NULL;

struct TextAttr TestFont = { /* Needed for opening screen. */
    (STRPTR)"topaz.font", TOPAZ_EIGHTY, 0, 0
};

/* DBL BUF */
struct BitMap *MyBitMapPtrs[2] = {NULL, NULL};
WORD ToggleFrame = 0;

struct GelsInfo GInfo;          /* For all Gels. */

struct VSprite *VSprites[NSPRITES];
WORDBITS VSpriteImage[] = {
/* Plane 0, Plane 1 */
    0xFFFF, 0xFFFF, /* Line 1, first. */
    0xFFFF, 0xC003,
    0xFFFF, 0xC003,
    0xF00F, 0xCFE3,
    0xF00F, 0xCFE3,
    0xF00F, 0xCC33,
    0xF00F, 0xCC33,
    0xF00F, 0xCFE3,
    0xF00F, 0xCFE3,
    0xFFFF, 0xC003,
    0xFFFF, 0xC003,
    0xFFFF, 0xFFFF, /* Line 12, last. */
};

```

```

USHORT *VSpriteImage_chip = 0;

/* These are the colors that will be used for my VSprites. Note I really do mean
 * colors, not color register numbers. High to low, starting at bit 12 and going
 * down to LSB, there are four bits each of red, green and blue. Please read the
 * sprite section of the hardware manual. The gels system will put them into the
 * proper color registers when they are displayed. Reminder: Sprites can only
 * use color registers in sets of 3...
 * 17,18,19 = sprite 0 and 1,
 * 21,22,23 = sprite 2 and 3,
 * 25,26,27 = sprite 4 and 5,
 * 29,30,31 = sprite 6 and 7.
 * Please read the section on how VSprites are assigned in the RKM.
 */
WORD MyVSpriteColors[] = {
    0x0f00, /* Full red. */
    0x00f0, /* Full green. */
    0x000f /* Full blue. */
};

struct Bob *Bobs[NBOBS];
short BobImage[] = {
    0xEFFF, 0xEFFF, 0xEFFF, 0xEFFC, /* Plane 0, line 1. */
    0xC000, 0x0000, 0x0000, 0x000C,
    0xCFFE, 0xEFFF, 0xEFFF, 0xEFFC,
    0xCC00, 0x0000, 0x0000, 0x00CC,
    0xCFFE, 0xEFFF, 0xEFFF, 0xEFFC,
    0xCCC0, 0x0000, 0x0000, 0x0CCC,
    0xCCCE, 0xEFFF, 0xEFFF, 0xCCCC,
    0xCCCC, 0x0000, 0x0000, 0xCCCC,
    0xCCCC, 0xEFFF, 0xEFFC, 0xCCCC,
    0xCCCC, 0xC000, 0x000C, 0xCCCC,
    0xCCCC, 0xCFFE, 0xEFFC, 0xCCCC,
    0xCCCC, 0xCC00, 0x00CC, 0xCCCC,
    0xCCCC, 0xCFFE, 0xFCCC, 0xCCCC,
    0xCCCC, 0xCCC0, 0x0CCC, 0xCCCC,
    0xCCCC, 0xCCCE, 0xCCCC, 0xCCCC,
    0xCCCC, 0xCCCC, 0xCCCC, 0xCCCC,
    0xCCCC, 0xCCCC, 0xCCCC, 0xCCCC,
    0xCCCC, 0xCCC0, 0x0CCC, 0xCCCC,
    0xCCCC, 0xCFFE, 0xEFFC, 0xCCCC,
    0xCCCC, 0xC000, 0x000C, 0xCCCC,
    0xCCCC, 0xEFFF, 0xEFFC, 0xCCCC,
    0xCCCC, 0x0000, 0x0000, 0xCCCC,
    0xCCCE, 0xEFFF, 0xEFFF, 0xCCCC,
    0xCCC0, 0x0000, 0x0000, 0x0CCC,
    0xCFFE, 0xEFFF, 0xEFFF, 0xEFFC,
    0xCC00, 0x0000, 0x0000, 0x00CC,
    0xCFFE, 0xEFFF, 0xEFFF, 0xEFFC,
    0xC000, 0x0000, 0x0000, 0x000C,
    0xEFFF, 0xEFFF, 0xEFFF, 0xEFFC, /* Plane 0, line 31. */

    0xEFFF, 0xEFFF, 0xEFFF, 0xEFFC, /* Plane 1, line 1. */
    0xEFFF, 0xEFFF, 0xEFFF, 0xEFFC,

    0xF000, 0x0000, 0x0000, 0x003C,
    0xF000, 0x0000, 0x0000, 0x003C,
    0xF0FE, 0xEFFF, 0xEFFF, 0xFC3C,
    0xF0FE, 0xEFFF, 0xEFFF, 0xFC3C,
    0xF0F0, 0x0000, 0x0000, 0x3C3C,
    0xF0F0, 0x0000, 0x0000, 0x3C3C,
    0xF0F0, 0xEFFF, 0xEFFC, 0x3C3C,
    0xF0F0, 0xEFFF, 0xEFFC, 0x3C3C,
    0xF0F0, 0xF000, 0x003C, 0x3C3C,
    0xF0F0, 0xF000, 0x003C, 0x3C3C,
    0xF0F0, 0xF0FE, 0xFC3C, 0x3C3C,
    0xF0F0, 0xF0FE, 0xFC3C, 0x3C3C,
    0xF0F0, 0xF0F0, 0x3C3C, 0x3C3C,
    0xF0F0, 0xF0F0, 0x3C3C, 0x3C3C,
    0xF0F0, 0xF0F0, 0x3C3C, 0x3C3C,
    0xF0F0, 0xF0FE, 0xFC3C, 0x3C3C,
    0xF0F0, 0xF0FE, 0xFC3C, 0x3C3C,
    0xF0F0, 0xF000, 0x003C, 0x3C3C,
    0xF0F0, 0xF000, 0x003C, 0x3C3C,
    0xF0F0, 0xEFFF, 0xEFFC, 0x3C3C,
    0xF0F0, 0xEFFF, 0xEFFC, 0x3C3C,
    0xF0F0, 0x0000, 0x0000, 0x3C3C,
    0xF0F0, 0x0000, 0x0000, 0x3C3C,
    0xF0FE, 0xEFFF, 0xEFFF, 0xFC3C,
    0xF0FE, 0xEFFF, 0xEFFF, 0xFC3C,
    0xF000, 0x0000, 0x0000, 0x003C,
    0xF000, 0x0000, 0x0000, 0x003C,
    0xEFFF, 0xEFFF, 0xEFFF, 0xEFFC, /* Plane 1, line 31. */

    0xEFFF, 0xEFFF, 0xEFFF, 0xEFFC, /* Plane 2, line 1. */
    0xEFFF, 0xEFFF, 0xEFFF, 0xEFFC,
    0xEFFF, 0xEFFF, 0xEFFF, 0xEFFC,
    0xFF00, 0x0000, 0x0000, 0x03FC,
    0xFF00, 0x0000, 0x0000, 0x03FC,
    0xFF00, 0x0000, 0x0000, 0x03FC,
    0xFF00, 0x0000, 0x0000, 0x03FC,
    0xFF00, 0xEFFF, 0xEFFC, 0x03FC,
    0xFF00, 0xEFFF, 0xEFFC, 0x03FC,
    0xFF00, 0xEFFF, 0xEFFC, 0x03FC,
    0xFF00, 0xEFFF, 0xEFFC, 0x03FC,
    0xFF00, 0xEFFF, 0xEFFC, 0x03FC,
    0xFF00, 0xEFFF, 0xEFFC, 0x03FC,
    0xFF00, 0xEFFF, 0xEFFC, 0x03FC,
    0xFF00, 0xEFFF, 0xEFFC, 0x03FC,
    0xFF00, 0xEFFF, 0xEFFC, 0x03FC,
    0xFF00, 0xEFFF, 0xEFFC, 0x03FC,
    0xFF00, 0xEFFF, 0xEFFC, 0x03FC,
    0xFF00, 0xEFFF, 0xEFFC, 0x03FC,
    0xFF00, 0xEFFF, 0xEFFC, 0x03FC,
    0xFF00, 0xEFFF, 0xEFFC, 0x03FC,
    0xFF00, 0xEFFF, 0xEFFC, 0x03FC,
    0xFF00, 0xEFFF, 0xEFFC, 0x03FC,
    0xFF00, 0xEFFF, 0xEFFC, 0x03FC,
    0xFF00, 0xEFFF, 0xEFFC, 0x03FC,
    0xFF00, 0x0000, 0x0000, 0x03FC,
    0xFF00, 0x0000, 0x0000, 0x03FC,
    0xFF00, 0x0000, 0x0000, 0x03FC,

```

```

0xEFF0, 0x0000, 0x0000, 0x03FC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC, /* Plane 2, line 31. */

0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC, /* Plane 3, line 1. */
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC,
0xEFFF, 0xEFFF, 0xEFFF, 0xFFFC, /* Plane 3, line 31. */
};
USHORT *BobImage_chip = 0;

/* These are for my custom screen. */
struct Screen *screen = NULL;
struct NewScreen ns = {
    0, 0, /* Start position. */
    SBMWIDTH, SBMHEIGHT, SBMDEPTH, /* Width, height, depth. */
    0, 0, /* Default detail pen, block pen. */
    NULL, /* Viewing mode. */
    CUSTOMSCREEN | CUSTOMBITMAP, /* Screen type. DBL BUF */
    &TestFont, /* Font to use. */
    NULL, /* No default title. */
    NULL, /* No pointer to additional gadgets. */
    NULL, /* No pointer to CustomBitMap. */
};

/* These are for my window. */
struct Window *window = NULL;

struct NewWindow nw = {
    0, 0, /* Start position. */
    SBMWIDTH, SBMHEIGHT, /* Width, height. */
    0, 0, /* Detail pen, block pen. */
    CLOSEWINDOW, /* IDCMP flags. */
    WINDOWCLOSE | BORDERLESS, /* Flags. */
    NULL, /* No pointer to FirstGadget. */
    NULL, /* No pointer to first CheckMark. */
    NULL, /* No default Title. */
    NULL, /* No pointer to Screen. */
    NULL, /* No pointer to BitMap. */
    0, 0, /* MinWidth, MinHeight (not used). */
    SBMWIDTH, SBMHEIGHT, /* MaxWidth, MaxHeight (not used). */
    CUSTOMSCREEN /* Screen type. */
};

/*****
* This will be called if a sprite collision with the border is detected.
*/

borderPatrol(s, b)
struct VSprite *s;
int b;
{
    register struct vInfo *info;

    info = &s->VUserExt;
    if (b & (TOPHIT | BOTTOMHIT)) /* Top/Bottom hit, change direction. */
        info->vy = -(info->vy);
    if (b & (LEFTHIT | RIGHTHIT)) /* Left/Right hit, change direction. */
        info->vx = -(info->vx);
}

/*****
* Fun Starts.
*/

main()
{
    SHORT i, j;

    /* Open libraries that will be used directly. */
    if ((IntuitionBase = (struct IntuitionBase *)
        OpenLibrary("intuition.library", LIBRARY_VERSION)) == 0) {
#ifdef DEBUG
        kprintf("Main: Can't open Intuition.\n");
#endif
        MyCleanup();
        Exit(-1);
    }

    if ((GfxBase = (struct GfxBase *)
        OpenLibrary("graphics.library", LIBRARY_VERSION)) == 0) {
#ifdef DEBUG
        kprintf("Main: Can't open Graphics.\n");

```

```

#endif
    MyCleanup();
    Exit(-1);
}

/*****
 * DBL BUF
 */
for (j=0; j<2; j++) {
    if ((MyBitMapPtrs[j] = (struct BitMap *)
        AllocMem(sizeof(struct BitMap), MEMF_CHIP)) == 0) {
#ifdef DEBUG
        kprintf("Main: Can't allocate BitMap.\n");
#endif
        MyCleanup();
        Exit(-1);
    }
    InitBitMap(MyBitMapPtrs[j], RBMDEPTH, RBMWIDTH, RBMHEIGHT);
    for (i=0; i<RBMDEPTH; i++) {
        if ((MyBitMapPtrs[j]->Planes[i] = (PLANEPTR)AllocRaster(RBMWIDTH,
            RBMHEIGHT)) == 0) {
#ifdef DEBUG
            kprintf("Main: Can't allocate BitMaps' Planes.\n");
#endif
            MyCleanup();
            Exit(-1);
        }
        BitClear(MyBitMapPtrs[j]->Planes[i], (RBMWIDTH / 8) * RBMHEIGHT, 1);
    }
    ns.CustomBitMap = MyBitMapPtrs[0]; /* !! */
    screen->RastPort.Flags = DBUEFFER;

    /* Open My Very Own Screen. */
    if ((screen = (struct Screen *)OpenScreen(&ns)) == 0) {
#ifdef DEBUG
        kprintf("Main: Can't open Screen.\n");
#endif
        MyCleanup();
        Exit(-1);
    }

    /* Now get that flashing title bar off the display. DBL BUF */
    screen->ViewPort.RasInfo->RxOffset = 5;
    screen->ViewPort.RasInfo->RyOffset = 5;
}

/* Set screens' colors (Could've used LoadRGB4()). */
SetRGB4(&screen->ViewPort, 00, 00, 00, 00);
SetRGB4(&screen->ViewPort, 01, 15, 00, 00);
SetRGB4(&screen->ViewPort, 02, 00, 15, 00);
SetRGB4(&screen->ViewPort, 03, 00, 00, 15);
SetRGB4(&screen->ViewPort, 04, 15, 15, 00);
SetRGB4(&screen->ViewPort, 05, 15, 00, 15);
SetRGB4(&screen->ViewPort, 06, 08, 15, 15);

SetRGB4(&screen->ViewPort, 07, 15, 11, 00);
SetRGB4(&screen->ViewPort, 08, 05, 13, 00);
SetRGB4(&screen->ViewPort, 09, 14, 03, 00);
SetRGB4(&screen->ViewPort, 10, 15, 02, 14);
SetRGB4(&screen->ViewPort, 11, 15, 13, 11);
SetRGB4(&screen->ViewPort, 12, 12, 09, 08);
SetRGB4(&screen->ViewPort, 13, 11, 11, 11);
SetRGB4(&screen->ViewPort, 14, 07, 13, 15);
SetRGB4(&screen->ViewPort, 15, 15, 15, 15);

nw.Screen = screen;

if ((window = (struct Window *)OpenWindow(&nw)) == 0) {
#ifdef DEBUG
    kprintf("Main: Can't open Window.\n");
#endif
    MyCleanup();
    Exit(-1);
}

/*****
 * Now that the screen environment is set up, It's time to set up the
 * gels system.
 */

/* ReadyGels is in GelTools(). */
if (ReadyGels(&GInfo, &screen->RastPort) != 0) {
#ifdef DEBUG
    kprintf("Main: ReadyGels failed.\n");
#endif
    MyCleanup();
    Exit(-1);
}

SetCollision(0, borderPatrol, &GInfo);

/* Copy Images to chip memory. */
if (!InitImages()) {
#ifdef DEBUG
    kprintf("Main: InitImages() failed.\n");
#endif
    MyCleanup();
    Exit(-1);
}

/*****
 * System is set up, now set up each Gel.
 */

/* First use the routines in geltools to get the sprite. */
for (i = 0; i < NSPRITES; i++) {
    if ((VSprite[i] = (struct VSprite *)MakeVSprite(VSPRITEHEIGHT,
        VSpriteImage_chip, &MyVSpriteColors[0], i*6, (i*8)+10,
        VSPRITEWIDTH, VSPRITEDEPTH, VSPRITE)) == 0) {
#ifdef DEBUG
        kprintf("Main: MakeVSprite failed.\n");
#endif

```



```

#endif
    MyCleanup();
    Exit(-1);
}

VSprites[i]->VUserExt.vx = 1;
VSprites[i]->VUserExt.vy = 1;
VSprites[i]->VUserExt.id = i;

AddVSprite(VSprites[i], &screen->RastPort);
}

/* First use the routines in geltools to get the bob. */
for(i = 0; i < NBOBS; i++) {
    if ((Bobs[i] = (struct Bob *)MakeBob(BOBWIDTH, BOBHHEIGHT, BOBDEPTH,
        BobImage_chip, 0x0F, 0x00, (i*6), (i*8)+10,
        SAVEBACK | OVERLAY)) == 0) {
#ifdef DEBUG
        kprintf("Main: MakeBob failed.\n");
#endif
        MyCleanup();
        Exit(-1);
    }

    Bobs[i]->BobVSprite->VUserExt.vx = 1;
    Bobs[i]->BobVSprite->VUserExt.vy = 1;
    Bobs[i]->BobVSprite->VUserExt.id = i;

    /* DBL BUF */
    if ((Bobs[i]->DBuffer = (struct DBufPacket *)AllocMem (sizeof(struct
        DBufPacket), MEME_CHIP)) == 0) {
#ifdef DEBUG
        kprintf("Main: Can't allocate double buffers' packet for a bob.\n");
#endif
        MyCleanup();
        Exit(-1);
    }

    if ((Bobs[i]->DBuffer->BufBuffer = (WORD *)AllocMem (sizeof(SHORT) *
        ((BOBWIDTH+15)/16) * BOBHHEIGHT * BOBDEPTH, MEME_CHIP)) == 0) {
#ifdef DEBUG
        kprintf("Main: Can't allocate double buffer for a bob.\n");
#endif
        MyCleanup();
        Exit(-1);
    }

    AddBob(Bobs[i], &screen->RastPort);

    /* The following relies on the fact that AddBob sets the before
    * and after pointers to 0, so the first before and last after
    * are left alone.
    * Earlier bob has higher priority, thus this bob'll be drawn
    * AFTER that one, thus this bob will appear on top of all earlier
    * ones. One could set the bobs to be drawn in any order by rearranging
    * these pointers.
    */
    if (i > 0) {

        Bobs[i]->After = Bobs[i-1];
        Bobs[i]->After->Before = Bobs[i];
    } /* End of for. */

    /******
    * Hey, wow, everything opened, and allocated, and initialized! Whew.
    */
    for (;;) {
        DrawGels();

        while (MyIntuiMessage = (struct IntuiMessage *)
            GetMsg(window->UserPort))
            switch (MyIntuiMessage->Class) {
                case CLOSEWINDOW:
                    ReplyMsg(MyIntuiMessage);
                    MyCleanup();
                    Exit(TRUE);
                    break;
                default:
                    ReplyMsg(MyIntuiMessage);
                    break;
            }
    }

    /******
    * DrawGels part of loop.
    */

    DrawGels()
    {
        register struct VSprite *pSprite;

        /* Move everything in the sprite list. This includes Bobs. */
        pSprite = GInfo.gelHead->NextVSprite;
        while (pSprite != GInfo.gelTail){
            pSprite->X += pSprite->VUserExt.vx;
            pSprite->Y += pSprite->VUserExt.vy;
            pSprite = pSprite->NextVSprite;
        }

        SortGList(&screen->RastPort); /* Put the list in order. */
        DoCollision(&screen->RastPort); /* Collision routines may called now. */
        DrawGList(&screen->RastPort, &screen->ViewPort); /* Draw 'em. */
        screen->ViewPort.RasInfo->BitMap = MyBitMapPtrs[ToggleFrame]; /* DBL BUF */
        WaitTOF(); /* When the beam hits the top... */
        MakeScreen(screen); /* Tell intuition to do it's stuff. */
        RethinkDisplay(); /* Does a MrgCop & LoadView. */
        ToggleFrame ^= 1; /* DBL BUF */
        screen->RastPort.BitMap = MyBitMapPtrs[ToggleFrame]; /* DBL BUF */
    }

    /******
    * This will be called in case of error, or when main is done.
    */
}

```

```

MyCleanup()
{
    short i, j;

    for (i=0; i < NBBOBS; i++) {
        if (Bobs[i] != NULL) {
            DeleteGel (Bobs[i]->BobVSprite);
        }
    }

    for (i=0; i < NSPRITES; i++) {
        if (VSprites[i] != NULL) {
            DeleteGel (VSprites[i]);
        }
    }
    PurgeGels (&GInfo);
    FreeImages ();
    if (window != NULL)
        CloseWindow(window);
    if (screen != NULL)
        CloseScreen(screen);

    /* DEL BUF */
    for (j=0; j<2; j++) {
        if (MyBitMapPtrs[j] != NULL) {
            for (i=0; i<RBMDEPTH; i++) {
                if (MyBitMapPtrs[j]->Planes[i] != 0)
                    FreeRaster (MyBitMapPtrs[j]->Planes[i], RBMWIDTH, RBMHEIGHT);
            }
            FreeMem(MyBitMapPtrs[j], sizeof(struct BitMap));
        }
    }

    if (GfxBase != NULL)
        CloseLibrary(GfxBase);
    if (IntuitionBase != NULL)
        CloseLibrary(IntuitionBase);
}

InitImages ()
{
    extern USHORT *VSpriteImage_chip;
    extern USHORT *BobImage_chip;
    int i;

    if ((VSpriteImage_chip = (USHORT *)
        AllocMem(sizeof(VSpriteImage), MEME_CHIP)) == 0) {
#ifdef DEBUG
        kprintf("InitImages: No Memory for VSpriteImage.\n");
#endif
        return(FALSE);
    }
    if ((BobImage_chip = (USHORT *)
        AllocMem(sizeof(BobImage), MEME_CHIP)) == 0) {
#ifdef DEBUG
        kprintf("InitImages: No Memory for BobImage.\n");
#endif
        return(FALSE);
    }
    for (i=0; i<24; i++)
        VSpriteImage_chip[i] = VSpriteImage[i];
    for (i=0; i<496; i++)
        BobImage_chip[i] = BobImage[i];
    return(TRUE);
}

FreeImages ()
{
    extern USHORT *VSpriteImage_chip;
    extern USHORT *BobImage_chip;

    if (VSpriteImage_chip != 0)
        FreeMem(VSpriteImage_chip, sizeof(VSpriteImage));
    if (BobImage_chip != 0)
        FreeMem(BobImage_chip, sizeof(BobImage));
}

```

```

/**** intuall.h *****/
*
* intuall.h, general includer for intuition
*
* Confidential Information: Commodore-Amiga, Inc.
* Copyright (c) Commodore-Amiga, Inc.
*
*
*      date      author :      Modification History
*      -----      -      -      -
*      1-30-85      --RJ--      created this file!
*
*****/

#include <exec/types.h>
#include <exec/nodes.h>
#include <exec/lists.h>
/* #include <exec/interrupts.h> */
#include <exec/memory.h>
#include <exec/ports.h>
#include <exec/tasks.h>
#include <exec/libraries.h>
#include <exec/devices.h>
#include <exec/io.h>
#include <exec/devices.h>

#include <devices/console.h>
#include <devices/timer.h>
#include <devices/keymap.h>
#include <devices/inpotevent.h>

#define Msg IOStcReq /* temporary kluge for dosextens.h */

#include <libraries/dos.h>
#include <libraries/dosextens.h>

#include <graphics/gfx.h> /* ALWAYS INCLUDE GEX.H before other includes */
#include <graphics/regions.h> /* new as of 7/9/85 */
#include <hardware/blit.h>

#define blitNode bltnode /* temporary kluge for gels.h */

#include <graphics/collide.h>
#include <graphics/copper.h>
#include <graphics/display.h>
#include <hardware/dmabits.h>
#include <graphics/gels.h>
#include <graphics/clip.h>
#include <graphics/rastport.h>
#include <graphics/view.h>
#include <graphics/gfxbase.h>
#include <graphics/text.h>
/* #include <hardware/intbits.h> */

#include <hardware/custom.h>
#include <graphics/gfxmacros.h>
#include <graphics/layers.h>
#include <intuition/intuition.h> /* changed so I can get gadget addr */
#include <devices/gameport.h>

```

```

/* =====
GELTOOLS.C -
A FILE CONTAINING USEFUL SETUP TOOLS FOR THE ANIMATION SYSTEM

author: Rob Peck, incorporating valuable comments and changes from
Barry Whitebook and David Lucas.
===== */

#include <exec/types.h>
#include <exec/memory.h>
#include <graphics/gfx.h> /* ALWAYS INCLUDE GFX.H before other includes */
#include <graphics/gels.h>
#include <graphics/clip.h>
#include <graphics/rastport.h>
#include <graphics/view.h>
#include <graphics/gfxbase.h>

```

```

/*****
* This file is a collection of tools which are used with the vsprite and
* bob software. It contains the following:
*
* ReadyGels( *gelsinfo, *rastport );
* PurgeGels( *gelsinfo );
*
* struct VSprite *MakeVSprite(lineheight,*image,*colorset,x,y,
* wordwidth,imagedepth,flags);
* DeleteVSprite( &VSprite );
*
* struct Bob *MakeBob(bitwidth,lineheight,imagedepth,*image,
* planePick,planeOnOff,x,y)
* DeleteBob( &Bob );
*
* ReadyGels sets up the defaults of the gel system by initializing the
* GelsInfo structure you provide. First it allocates room for and
* links in lastcolor and nextline. It then uses information in your
* RastPort structure to establish boundary collision defaults at
* the outer edges of the raster. It then links together the GelsInfo
* and the RastPort which you provide. Next it allocates space for two
* dummy virtual sprite structures, calls InitGels and SetCollision.
* You must already have run LoadView before ReadyGels is called.
*
* PurgeGels deallocates all memory which ReadyGels and NewCellList have
* allocated. The system will crash if you have not used these
* routines to allocate the space (you cant deallocate something
* which you havent allocated in the first place).
*
* MakeVSprite allocates enough space for and inits a normal vsprite.
* DeleteVSprite deallocates the memory it used.
*
* MakeBob initializes a standard bob and allocates as much memory as is needed
* for a normal bob and its vsprite structure, links them together.
* To find the associated vsprite, look at the back-pointer (see the
* routine doc itself).
* DeleteBob deallocates the memory it used.
*
* Written by Rob Peck, with thanks to Barry Whitebrook and David Lucas.
*
*****/

```

```

void border_dummy()
{
    return;
}

/* Caller passes a pointer to his GelsInfo structure which he wants to init,
 * along with a pointer to his IVPArgs. Default init places the topmost
 * bottommost etc at the outermost boundaries of callers rastport parameters.
 * Caller can change all this stuff after this routine returns.
 */

extern struct RastPort *myRast;

struct VSprite *SpriteHead = NULL;
struct VSprite *SpriteTail = NULL;

/*****
 * This routine cannot be run until the first LoadView(&view) has been
 * executed. InitGels works with an already active View, so LoadView
 * must have been run first.
 */

ReadyGels(g, r)
struct RastPort *r;
struct GelsInfo *g;
{
    /* Allocate head and tail of list. */
    if ((SpriteHead = (struct VSprite *)AllocMem(sizeof
(struct VSprite), MEMF_PUBLIC | MEMF_CLEAR)) == 0) {
#ifdef DEBUG
        kprintf("ReadyGels: No memory for sprite head.\n");
#endif
        return(-1);
    }

    if ((SpriteTail = (struct VSprite *)AllocMem(sizeof
(struct VSprite), MEMF_PUBLIC | MEMF_CLEAR)) == 0) {
#ifdef DEBUG
        kprintf("ReadyGels: No memory for sprite tail.\n");
#endif
        return(-1);
    }

    /* By setting all bits here, it means that there are NO
     * reserved sprites. The system can freely use all of the
     * hardware sprites for its own purposes. The caller will not be
     * trying to independently use any hardware sprites!
     */
    g->sprRsrvd = -1;

    /* The nextline array is used to hold system information about
     * "at which line number on the screen is this hardware sprite
     * again going to become available to be given a new vsprite to
     * display".
     */

    if ((g->nextLine = (WORD *)AllocMem(sizeof(WORD) * 8,
MEMF_PUBLIC | MEMF_CLEAR)) == NULL) {
#ifdef DEBUG
        kprintf("ReadyGels: No memory for nextline.\n");
#endif
        return(-1);
    }

    /* In the lastcolor pointer array, the system will store
     * a pointer to the color definitions most recently used
     * by the system. .... as a reminder, virtual sprites can
     * be assigned to any of the real hardware sprites which
     * may be available at the time. The vsprite colors will
     * be written into the hardware sprite register set for
     * the hardware sprite to which that vsprite is assigned.
     * This pointer array contains one pointer to the last
     * set of three colors (from the vsprite structure *sprColors)
     * for each hardware sprite.
     *
     * As the system is scanning to determine which hardware
     * sprite should next be used to represent a vsprite, it
     * checks the contents of this array. If a hardware sprite
     * is available and already has been assigned this set of
     * colors, no color assignment is needed, and therefore
     * no color change instructions will be generated for the
     * copper list.
     *
     * If all vsprites use a different set of sprColors, (pointers
     * to sprColors are different for all vsprites), then there
     * is a limit of 4 vsprites on a horizontal line. If, on
     * the other hand, you define, lets say 8 vsprites, with
     * 1 and 2 having the same sprColors, 3 and 4 the same as
     * each other, 5 and 6 the same as each other, and 7 and 8
     * also having the same vsprite colors, then you will be
     * able to have all 8 vsprites on the same horizontal line.
     *
     * In this case, you will be able to put all 8 vsprites on
     * the same horizontal line. The reason this helps is that
     * the system hardware shares the color registers between pairs
     * of hardware sprites. The system thus has enough resources
     * to assign all vsprites to hardware sprites in that there
     * are 4 color-sets for 8 vsprites, exactly matching the
     * hardware maximum capabilities.
     *
     * Note that lastcolor will not be used for bobs. Just sprites.
     */
    if ((g->lastColor = (WORD **)AllocMem(sizeof(LONG) * 8,
MEMF_PUBLIC | MEMF_CLEAR)) == NULL) {
#ifdef DEBUG
        kprintf("ReadyGels: No memory for lastcolor.\n");
#endif
        return(-1);
    }

    /* This is a table of pointers to the routines which should
     * be performed when DoCollision senses a collision. This

```

```

* declaration may not be necessary for a basic vsprite with
* no collision detection implemented, but then it makes for
* a complete example.
*/
if ((g->collHandler = (struct collTable *)AllocMem(sizeof(struct
collTable), MEME_PUBLIC | MEME_CLEAR)) == NULL) {
#ifdef DEBUG
    kprintf("ReadyGels: No memory for collHandler.\n");
#endif
    return(-1);
}

/* When any part of the object touches or passes across
* this boundary, it will cause the boundary collision
* routine to be called. This is at smash[0] in the
* collision handler table and is called only if
* DoCollision is called.
*/
g->leftmost = 0;
g->rightmost = r->BitMap->BytesPerRow * 8 - 1;
g->topmost = 0;
g->bottommost = r->BitMap->Rows - 1;

r->GelsInfo = g; /* Link together the two structures */
InitGels(SpriteHead, SpriteTail, g);

/* Pointers initialized to the dummy sprites which will be
* used by the system to keep track of the animation system.
*/
SetCollision(0, border_dummy, g);
WaitTOF();
return(0);
}

/*****
* Use this to get rid of the gels stuff when it is not needed any more.
* You must have allocated the gels info stuff (use the ReadyGels routine).
*/

PurgeGels(g)
struct GelsInfo *g;
{
    if (g->collHandler != NULL)
        FreeMem(g->collHandler, sizeof(struct collTable));
    if (g->lastColor != NULL)
        FreeMem(g->lastColor, sizeof(LONG) * 8);
    if (g->nextLine != NULL)
        FreeMem(g->nextLine, sizeof(WORD) * 8);
    if (g->gelHead != NULL)
        FreeMem(g->gelHead, sizeof(struct VSprite));
    if (g->gelTail != NULL)
        FreeMem(g->gelTail, sizeof(struct VSprite));
}

/*****

* Because MakeVSprite is called by MakeBob, MakeVSprite only creates the
* VSprite, it doesn't add it to the system list. The calling routine must
* do an AddVSprite after it is created.
*/

struct VSprite *MakeVSprite(lineheight, image, colorset, x, y,
                            wordwidth, imagedepth, flags)
SHORT lineheight; /* How tall is this vsprite? */
WORD *image; /* Where is the vsprite image data, should be
              twice as many words as the value of lineheight */
WORD *colorset; /* Where is the set of three words which describes
                the colors that this vsprite can take on? */
SHORT x, y; /* What is its initial onscreen position? */
SHORT wordwidth, imagedepth, flags;
{
    struct VSprite *v; /* Make a pointer to the vsprite structure which
                       this routine dynamically allocates */

    if ((v = (struct VSprite *)AllocMem(sizeof(struct VSprite),
MEME_PUBLIC | MEME_CLEAR)) == 0) {
#ifdef DEBUG
        printf("MakeVSprite: Couldn't allocate VSprite.\n");
#endif
        return(0);
    }

    v->Flags = flags; /* Is this a vsprite, not a bob? */

    v->Y = y; /* Establish initial position relative to */
    v->X = x; /* the Display coordinates. */

    v->Height = lineheight; /* The Caller says how high it is. */
    v->Width = wordwidth; /* A vsprite is always 1 word (16 bits) wide. */

    /* There are two kinds of depth... the depth of the image itself, and the
    * depth of the playfield into which it will be drawn. The image depth
    * says how much data space will be needed to store an image if it's
    * dynamically allocated. The playfield depth establishes how much space
    * will be needed to save and restore the background when a bob is drawn.
    * A vsprite is always 2 planes deep, but if it's being used to make a
    * bob, it may be deeper...
    */

    v->Depth = imagedepth;

    /* Assume that the caller at least has a default boundary collision
    * routine... bit 1 of this mask is reserved for boundary collision
    * detect during DoCollision(). The only collisions reported will be
    * with the borders. The caller can change all this later.
    */

    v->MeMask = 1;
    v->HitMask = 1;

    v->ImageData = image; /* Caller says where to find the image. */

```

```

/* Show system where to find a mask which is a squished down version
 * of the vsprite (allows for fast horizontal border collision detect).
 */
if ((v->BorderLine = (WORD *)AllocMem(sizeof(WORD)*wordwidth,
MEME_PUBLIC | MEME_CLEAR)) == 0) {
#ifdef DEBUG
kprintf("MakeVSprite: Couldn't allocate BorderLine.\n");
#endif
return(0);
}

/* Show system where to find the mask which contains a 1 bit for any
 * position in the object in any plane where there is a 1 bit (all planes
 * OR'ed together).
 */
if ((v->CollMask = (WORD *)AllocMem(sizeof(WORD)*lineheight*wordwidth,
MEME_CHIP | MEME_CLEAR)) == 0) {
#ifdef DEBUG
kprintf("MakeVSprite: Couldn't allocate CollMask.\n");
#endif
return(0);
}

/* This isn't used for a Bob, just a VSprite. It's where the
 * Caller says where to find the VSprites colors.
 */
v->SprColors = colorset;

/* These aren't used for a VSprite, and MakeBob'll do set up for Bob. */
v->PlanePick = 0x00;
v->PlaneOnOff = 0x00;

InitMasks(v); /* Create the collMask and borderLine */
return(v);
}

struct Bob *MakeBob(bitwidth, lineheight, imagedepth, image,
planePick, planeOnOff, x, y, flags)
SHORT bitwidth, lineheight, imagedepth, planePick, planeOnOff, x, y, flags;
WORD *image;
{
struct Bob *b;
struct VSprite *v;
SHORT wordwidth;

wordwidth = (bitwidth+15)/16;

/* Create a vsprite for this bob, it will need to be deallocated
 * later (freed) when this bob gets deleted.
 * Note: No color set for bobs.
 */
if ((v = MakeVSprite(lineheight, image, NULL, x, y, wordwidth,
imagedepth, flags)) == 0) {
#ifdef DEBUG
kprintf("MakeBob: MakeVSprite failed.\n");
#endif
return(0);
}

/* Caller selects which bit planes into which the image is drawn. */
v->PlanePick = planePick;

/* What happens to the bit planes into which the image is not drawn. */
v->PlaneOnOff = planeOnOff;

if ((b = (struct Bob *)AllocMem(sizeof(struct Bob),
MEME_PUBLIC | MEME_CLEAR)) == 0) {
#ifdef DEBUG
kprintf("MakeBob: Couldn't allocate bob.\n");
#endif
return(0);
}

v->VSBob = b; /* Link together the bob and its vsprite structures */
b->Flags = 0; /* Not part of an animation (BOBISCOMP) and don't keep the
image present after bob is removed (SAVEBOB) */

/* Tell where to save background. Must have enough space for as many
 * bitplanes deep as the display into which everything is being drawn.
 */
if ((b->SaveBuffer = (WORD *)AllocMem(sizeof(SHORT) * wordwidth
* lineheight * imagedepth, MEME_CHIP | MEME_CLEAR)) == 0) {
#ifdef DEBUG
kprintf("MakeBob: Couldn't allocate save buffer.\n");
#endif
return(0);
}

b->ImageShadow = v->CollMask;

/* Interbob priorities are set such that the earliest defined bobs have
 * the lowest priority, last bob defined is on top.
 */
b->Before = NULL; /* Let the caller worry about priority later. */
b->After = NULL;

b->BobVSprite = v;

/* InitMasks does not preset the imageShadow ... caller may elect to use
 * the collMask or to create his own version of a shadow, although it
 * is usually the same.
 */
b->BobComp = NULL; /* this is not part of an animation */
b->DBuffer = NULL; /* this is not double buffered */

/* Return a pointer to this newly created bob for additional caller

```

```

    * interaction or for AddBob(b);
    */
    return(b);
}

/* Deallocate memory which has been allocated by the routines Makexxx. */
/* Assumes images and imageshadow deallocated elsewhere. */
DeleteCel(v)
struct VSprite *v;
{
    if (v != NULL) {
        if (v->VSBob != NULL) {
            if (v->VSBob->SaveBuffer != NULL) {
                FreeMem(v->VSBob->SaveBuffer, sizeof(SHORT) * v->Width
                    * v->Height * v->Depth);
            }
            if (v->VSBob->DBuffer != NULL) {
                if (v->VSBob->DBuffer->BufBuffer != 0) {
                    FreeMem(v->VSBob->DBuffer->BufBuffer,
                        sizeof(SHORT) * v->Width * v->Height * v->Depth);
                }
                FreeMem(v->VSBob->DBuffer, sizeof(struct DBufPacket));
            }
            FreeMem(v->VSBob, sizeof(struct Bob));
        }
        if (v->CollMask != NULL) {
            FreeMem(v->CollMask, sizeof(WORD) * v->Height * v->Width);
        }
        if (v->BorderLine != NULL) {
            FreeMem(v->BorderLine, sizeof(WORD) * v->Width);
        }
        FreeMem(v, sizeof(struct VSprite));
    }
}

```


Chapter 4

Text

Introduction

Text on the Amiga is simply another graphics primitive. Because of this, you can easily intermix text and graphics on the same screen. Typically, a 320-by-200 graphics screen can contain 40-column, 25-line text using a text font defined in an 8-by-8 matrix. The same type of font can be used to display 80-column text if the screen resolution is extended to 640 by 200. Window borders and other graphics embellishments may reduce the actual available area.

The text support routines use the **RastPort** structure to hold the variables that control the text drawing process. Therefore, any changes you make to **RastPort** variables affect both the drawing routines and the text routines.

In addition to the basic fonts provided in the ROMs, you can link your own font into the system, and ask that it be used along with the other system fonts.

This chapter shows you how to:

- o Print text into a drawing area
- o Specify the character color
- o Specify which font to use
- o Access disk-based fonts
- o Link in a new font
- o Define a new font
- o Define a disk-based font

Printing Text into a Drawing Area

The placement of text in the drawing area depends on several variables. Among these are the current position for drawing operations, the font width and height, and the placement of the font baseline within that height.

CURSOR POSITION

Text position and drawing position use the same variables in the **RastPort** structure—**cp_y** and **cp_x**, the current vertical and horizontal pen position. The text character begins at this point. You use the graphics call **Move(&rastPort, x, y)** to establish the **cp_y** and **cp_x** position.

BASELINE OF THE TEXT

The **cp_y** position of the drawing pen specifies the position of the baseline of the text. In other words, all text printed into a **RastPort** using a single “write string” command is positioned relative to this **cp_y** as the text baseline. Figure 4-1 shows some sample text that includes a character that has 1 dot below the baseline and a maximum of 7 dots above and including the baseline.

For clarity, blank squares and shaded squares, rather than 0s and 1s, are used for the figure.

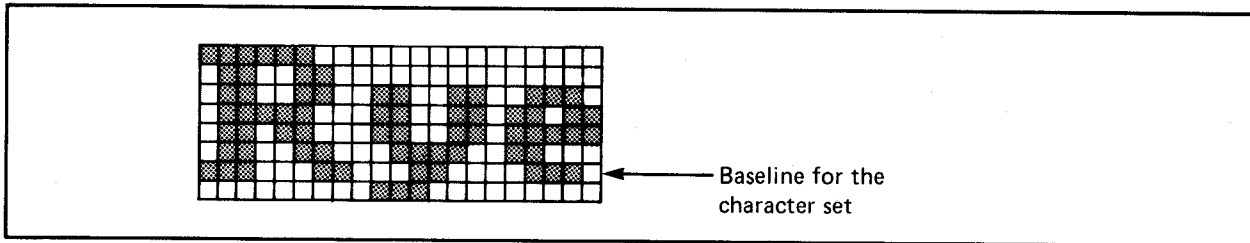


Figure 4-1: Text Baseline

The figure shows that for this font, the baseline value is 6. The baseline value is the number of lines from the top of the character to the baseline.

When the text routines output a character to a **RastPort**, the leftmost edge of the character position is specified by the **cp_x** (current horizontal position) variable.

After all characters have been written to the **RastPort**, the variable **cp_y** is unchanged. The value of **cp_x** will be changed by the number of horizontal positions that were needed to write all characters of the specified text. Both fixed-width and proportionally spaced character sets are accommodated.

The default fonts in the system are all designed to be above and below the baseline, where the baseline position is at line 6 of the character font. This means that you must specify a **cp_y** value of at least 6 when you request that text be printed to a **RastPort** in order to assure that you stay within the memory bounds of the **RastPort** itself. Location (0,0) specifies the upper left-hand corner of the memory space that is dedicated to the **RastPort**. Because all text will be written above and below the baseline, you must start at a proper position or the routines will write into non-**RastPort** memory.

You should not request that the text routines write beyond the outer bounds of the **RastPort** memory, either horizontally or vertically. Text written outside the **RastPort** bounds may be clipped if the **RastPort** supports clipping (most do). Clipping means that the system will display only that portion of the text that is written into the boundaries of the **RastPort**.

SIZE OF THE FONT

Font design is covered later in this chapter. For now, simply note that the width and height of the font affect how many characters you may print on a line. The position of the baseline affects where you print a line.

PRINTING THE TEXT

You may print text into a **RastPort** by using the **Text()** routine. A typical call to this routine is:

```
Text( &rastPort, string, count )
```

where

&rastPort is a pointer that describes where the text is to be output

string is the address of the string output

count is the string length

SAMPLE PRINT ROUTINE

Here is an example showing a string to be written to a **RastPort**. This example assumes that you have already prepared a **RastPort** into which the text can be rendered.

```
/* sample routine to print a single line of text to the screen. */
struct RastPort *rp;
test( )
{
SetAPen( rp, 1); /* use color number 1 to draw the text */
Move( rp, 0, 40); /* start down a few lines from the top */
Text( rp, "This is test text", 17 );
return();
}
```

Selecting the Font

Character fonts each have a name. Two default character fonts are provided in the ROMs. One font produces either 40- or 80-column text (depending on the use of a 320 or 640 horizontal resolution, respectively). The other font produces either 32- or 64-column text. The names and specifications of these default fonts are shown in table 4-1.

Table 4-1: Default Character Fonts

Font Type	Height	Name
40/80	8	topaz.font
32/64	9	topaz.font

To specify which font the system should use, you call the system routine **OpenFont()** or **OpenDiskFont()**, followed by **SetFont()**. A typical call to these routines follows.

```
font=OpenFont(textattr);  
font=OpenDiskFont(textattr);  
SetFont( font, rp )
```

where

font

is a pointer to a **TextFont** data structure, returned by either **OpenFont()** or **OpenDiskFont()**.

textattr

is a structure located in the include file *graphics/text.h*. It contains a pointer to a null-terminated string that specifies the name of the font, font height, font style bits, and font preference bits.

rp is the address of the **RastPort** that is to use that font until told to use a different one.

The call to **OpenFont()** or **OpenDiskFont()** says “give me a font with these characteristics.” The system attempts to fulfill your request by providing the font whose characteristics best match your request. The table above shows that both of the system fonts have the name “topaz.font.” In the system font selections, the height of the characters distinguishes between them. If **OpenFont()** cannot be satisfied, it returns a 0.

Note: In chapter 1, “Graphics Primitives,” you saw that the routine `InitRastPort()` initializes certain variables to default values. This routine automatically sets the default to `topaz.font` with the correct width according to Preferences.

The example below shows how a new font is selected. This example prints two lines of text to the screen, each line of text in a different font. It assumes that a `RastPort` is already set up elsewhere.

```
#include "graphics/text.h"

test( )
{
    struct TextAttr f;
        /* provide a font structure to build on for font change */
    struct TextFont *font;
    f.ta_Name = "topaz.font";
        /* set font name into font descriptor struct */
        /* initial font default is "topaz.font" */
    f.ta_YSize = 8;
        /* define font size */
    f.ta_Style = 0;
        /* define font style */
    f.ta_Flags = 0;
        /* define font preferences */
    font=OpenFont(&f);
    if (font !=0) {
        SetFont( rp, font);
            /* ask system to find & set one like this */
        Move( rp, 0, 40);
        Text( rp, "topaz.font, 8 dots high", 23 );
        CloseFont(font);
    }
    f.ta_Ysize=9;
    font=OpenFont(&f);
    if (font != 0) {
        SetFont(rp,font);
        Move( rp, 0, 48);
            /* start a few lines down from the top */
        Text( rp, "topaz.font, 9 dots high", 23);
        CloseFont(font);
    }
    return(0);
}
```

Selecting the Text Color

You can select which color to use for the text you print by using the graphics calls **SetAPen()** and **SetBPen()** and by selecting the drawing mode in your **RastPort** structure. The combination of those values determines exactly how the text will be printed.

Selecting a Drawing Mode

The **DrawMode** variable of a **RastPort** determines how the text will be combined with the graphics in the destination area.

Note: The **DrawMode** selections are *values*, not bits. You can select from any *one* of the following drawing modes.

If **DrawMode** is **JAM1**, it means that the text will be drawn in the color of **FgPen** (the foreground, or primary, drawing pen). Wherever there is a 1-bit in the text pattern, the **FgPen** color will overwrite the data present at the text position in the **RastPort**. This is called overstrike mode.

If **DrawMode** is **JAM2**, it means that the **FgPen** color will be used for the text, and the **BgPen** color (the background or secondary drawing color pen) will be used as the background color for the text. The rectangle of data bits that defines the text-character completely overlays the destination area in your **RastPort**. Where there is a 1 bit in the character pattern definition, the **FgPen** color is used. Where there is a 0 bit in the pattern, the **BgPen** color is used. This mode draws text with a colored background.

If **DrawMode** is **COMPLEMENT**, it means that wherever the text character is drawn, a position occupied by a 1 bit causes bits in the destination **RastPort** to be changed as follows (see also figure 4-2):

- o If a text-character 1 bit is to be written over a destination area 0 bit, it changes the destination area to a 1 bit.
- o If a text-character 1 bit is to be written over a destination area 1 bit, the result of combining the source and destination is a 0 bit. In other words, whatever the current state of a destination area bit, a 1 bit in the source changes it to the opposite state.
- o Zero bits in the text character definition have no effect on the destination area.

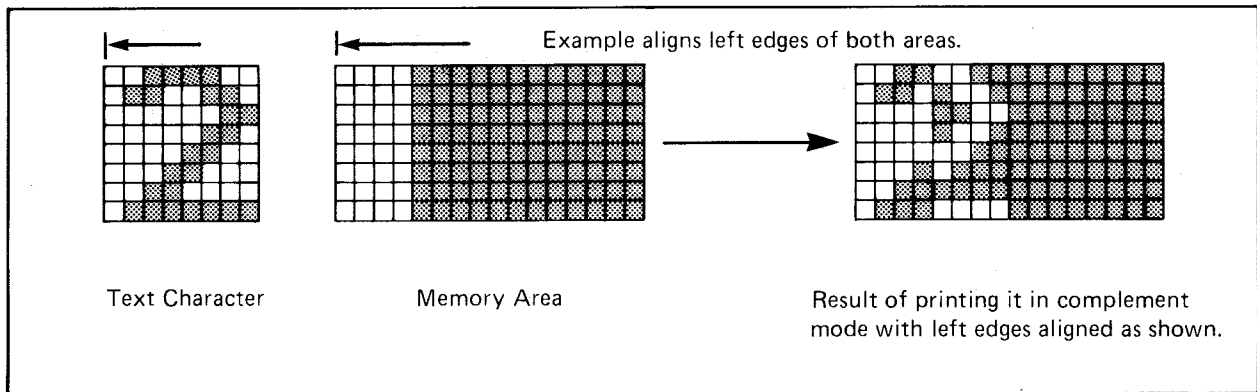


Figure 4-2: Complement Mode

If you set the **INVERSVID** flag to a 1, it will change all 1 bits to 0 bits and vice versa in a text or other **RastPort** writing operation before writing them into the destination area. If the drawing mode at that time is **JAM2**, then the pattern colors will be reversed as well. If **DrawMode** is **INVERSVID**, you can produce inverse video characters.

Here is an example showing each of the three modes of text that you can produce. Again it assumes that your **RastPort** has been set up elsewhere.

```

/* sample routine to print four lines of text to
 * the screen, each line in a different mode */
test()
{
  SetAPen( rp, 2);          /* use color 2 as primary drawing color */
  SetBPen( rp, 3);          /* use color 3 as secondary drawing color */
  Move( rp, 0, 6);         /* move the drawing position near upper left */
  SetDrMd( rp, JAM1 );     /* Jam 1 color into target raster */
  Text( rp, "This is JAM1 mode", 17 );
  Move( rp, 0, 46);        /* move the drawing position for next line */
  SetDrMd( rp, JAM2 );     /* Jam 2 colors into target raster */
  Text( rp, "This is JAM2 mode", 17 );
  Move( rp, 0, 86);        /* move the drawing position for next line */
  /* use exclusive-or (COMPLEMENT) to write */
  SetDrMd( rp, COMPLEMENT );
  Text( rp, "This is COMPLEMENT mode", 23 );
  Move( rp, 0, 126 );
  SetDrMd( rp, JAM1+INVERSEVID);
  Text( rp, "INVERSE", 7 );
  return;
}

```

Effects of Specifying Font Style

When you call **OpenFont()**, specifying certain style characteristics, the system searches the loaded fonts to find the closest match to the font you requested. If the remainder of the characteristics match what you have requested, but the style does not match, the text routines **AskSoftStyle()** and **SetSoftStyle()** create a font styled as you have requested by modifying the existing font (that is, modifying a normal font to italic or bold by modifying its characters.) Because many fonts do not lend themselves to such modifications, it is always preferred that the font of the specific style be loaded for use. The system always tries to find the exact specified font before attempting to modify another to fit your request.

If there is a font present in the system that matches your **OpenFont()** request both in name and size, but not in style, (as determined by looking at the font style field), you may use **SetSoftStyle()** to generate the selected style algorithmically as follows:

NORMAL

The font is used exactly as defined.

UNDERLINED

An underline is generated one pixel below the baseline position.

ITALIC

The character is given a slant to the right, starting from the bottom line, and shifting subsequent upward line positions to the right one bit position for every second count up from the bottom of the character.

EXTENDED

This attribute cannot be set with **SetSoftStyle()**. See "Font Style" below.

If you use a font that has the various style characteristics built in, rather than generated, the internal spacing and kerning tables tell the system how to leave the proper amount of space between characters if you are simply printing them one at a time.

If you ask **Text()** to output the characters individually, **Text()** calculates character positioning and width based on the *normal* width and inter-character spacing that it finds in the font descriptor. After printing one or more characters, it automatically positions the drawing pen (**cp_x**) at the position it believes to be correct for the next output character. This may cause adjacent characters to overlap when printed individually.

There is a solution to this problem. If you are using generated style for a font, you must take care to build your output strings of characters before calling **Text()** to output them. **Text()** can handle character strings, correctly generating the desired style with correct inter-character spacing.

To increase inter-character spacing, you can set a field called `rp_TxSpacing` in the `RastPort`. The spacing is specified in pixels.

Adding a New Font to the System

The ROM Exec code maintains a list of the text fonts that are currently linked into the system. To add another font, you must open a disk font using the diskfont library or define the font. You must also reserve some memory where the font can be loaded, move the font definition into that memory area, and link the font name and location into the system font list.

Using a Disk Font

To use an existing disk font, you must open the diskfont library and open a disk font. Here are the program fragments you need to open the library. This gives you access to whatever routines the diskfont library contains:

```
struct Library *DiskfontBase;

DiskfontBase = (struct Library *)
    OpenLibrary("diskfont.library",0);
```

Before trying to use the diskfont routines, you should check that the `OpenLibrary()` call returned a value other than `NULL`.

Here is the program fragment you need to actually load a disk-based font. It assumes that you already know the name of the font you want to load.

```
struct TextFont *font;
struct TextAttr myTextAttr;

font = OpenDiskFont(&myTextAttr);
```

Finding Out Which Fonts Are Available

The function **AvailFonts()** fills in a memory area designated by you to hold a list of all of the fonts available in the entire system. **AvailFonts()** searches the AmigaDOS directory path currently assigned to **FONTSD:** and locates all available fonts. If you haven't issued a **DOS ASSIGN** command to change the **FONTSD:** directory path, the system will search the *sys:fonts* directory.

The test program "whichfont.c" at the end of this chapter provides a list of the fonts you can use and shows you how to find the appropriate items to put into the text attribute data structure for the call to **OpenDiskFont()**.

Contents of a Font Directory

In a font directory, you will usually find two names for each font type. A typical pair of entries in the fonts directory is as follows:

```
sapphire.font  
sapphire(dir)
```

The file named *sapphire.font* does not contain the actual font. It contains the description of the contents of that font family. The contents are described by a **FontContentsHeader** and one or more **FontContents** data structure entries. The **FontContentsHeader** structure is defined in *libraries/diskfont.h* as:

```
struct FontContentsHeader {  
    UWORD fch_FileID;    /* FCH_ID */  
    UWORD fch_NumEntries; /* the number of FontContents elements */  
                        /* FontContents (1 or more) follow here */  
};
```

where

fch_FileID

is simply a numeric identifier for this file type. The value is 0xf00.

fch_NumEntries

says how many entries of type **FontContents** follows this header.

The **FontContents** structure is defined as follows:

```
struct FontContents {
    char    fc_FileName[MAXFONTPATH];
    UWORD   fc_YSize;
    UBYTE   fc_Style;
    UBYTE   fc_Flags;
};
```

where

fc_FileName

is the pathname that AmigaDOS must follow to find the actual diskfont descriptive header, along with the **TextFont** data structure of which this font is composed. Once AmigaDOS reaches the path named in **FONTS**:, it finds the filename by the path shown in this entry in **FontContents**.

fc_YSize, fc_Style, and fc_Flags

correspond to their equivalents in the **TextAttr** data structure (**ta_YSize, ta_Style, and ta_Flags**).

As an example, a typical entry in **sapphire.font** is:

"sapphire/14",	a null-terminated string, padded out with zeros for a length of MAXFONTPATH bytes,
14,	the value for fc_YSize,
00,	the value for fc_Style,
60 (hex)	the value for fc_Flags.

This entry indicates that the actual **DiskFontHeader** for the font to be loaded is in path **FONTS:sapphire/14**. This means that the **sapphire** subdirectory in the **fonts** directory must have a file named *14* in order to allow this font to be loaded.

The Disk Font

A disk font is constructed as a loadable, executable module. In this manner, AmigaDOS can be used to perform **LoadSegment()** and **UnloadSegment()** on it. AmigaDOS can therefore allocate memory for the font, and return the memory when the font is unloaded. The contents of the **DiskFont** are described in the include-file *libraries/diskfont.h*. The most significant item in this structure, the embedded **TextFont** structure, is described below in the topic "Defining a Font."

Defining a Font

To define a font, you must specify its characteristics using the **TextFont** structure. The **TextFont** structure is specified in the include file named *graphics/text.h*. The following topics show the meaning of the items in a **TextFont** structure. Following the structure description is an example showing a four-character font, which is defined using this structure and can be linked into the system using **AddFont()**.

THE TEXT NODE

The first item in the **TextFont** structure is a **listNode** by which the system can link this font structure into the system **TextFonts** list. You specify the name of the font using the name pointer field of the font **listNode**.

For example:

```
struct TextFont suitFont;
    /* name chosen for sample font here */
    suitFont.textNode.ln_name = "suits.font";
```

FONT HEIGHT

You specify the height in the **ySize** variable. All characters of the font must be defined using this number of lines of data even if they do not require that many lines to contain all font data. Variable-height fonts are not supported.

For example:

```
suitFont.ySize = 8; /* all characters are 8 lines high */
```

FONT STYLE

You can specify the style of the font by specifying certain bits as 1s in the **TextFont Style** variable. The value of **Style** is determined by the sum of the style bits, defined as:

NORMAL (value = 0),	The text font is used exactly as defined.
UNDERLINED (value = 1),	The font is underlined.
BOLD (value = 2),	The font is bold.
ITALIC (value = 4),	The font is italic.
EXTENDED (value = 8),	The font is stretched out (width).

In the font structure, these bits indicate style attributes as intrinsically a part of the font; that is, the font already has them and you can never take them away.

FONT PREFERENCES

This variable provides additional information that tells the font routines how to create or access the characters. The Preferences variable is composed of the sum of the preference bits, defined as follows:

FPB_ROMFONT (value = 0)

The font is located in ROM. If you are making up your own font, this variable will not be zero unless you are burning new system ROMs yourself.

FPB_REVPATH (value = 2)

The font is designed to be rendered from right to left (for example, Hebrew).

FPB_PROPORTIONAL (value = 32)

The characters in the font are not guaranteed to be **xSize** wide (see “Font Width” below). Each character has its own width and positioning in the character space. The bit-packing of the characters is of great importance, as described below. The variables **modulo**, **charloc**, and **charspace** define how the characters are defined and bit-packed.

FONT WIDTH

The **xSize** variable specifies the nominal width of the font. For example:

```
suitFont.tf_XSize = 14; /* specify 14 bits width */
```

FONT ACCESSORS

If you have added a font to the system list, it is possible that more than one task will be accessing a character font. A variable in the font structure keeps track of how many accessors this font currently has. Whenever you call **OpenFont()** or **OpenDiskFont()**, this variable is incremented for the font and decremented by **CloseFont()**. The font accessor value should never be reduced below zero. This accessor count should be initialized to zero *before* you first link a new font into the system, but it is managed by the system after the link is performed.

If you wish to remove a font from the system to free the memory that it is currently using, you must ensure that the number of accessors is zero before ordering its removal.

CHARACTERS REPRESENTED BY THIS FONT

It is possible to create a font consisting of 0 to 255 characters. Some fonts can be exceedingly large because of their design and the size of the characters. For this reason, the text system allows the design and loading of fonts that may consist of only a few of the characters. The variables **tf_loChar** and **tf_hiChar** specify the numerical values for the characters represented in this font. As an example, one font could contain only the capital letters. A second font could contain the small letters, and a third could contain the punctuation marks and numerals. Depending on the size of the font itself, you may arrange to subdivide the font even further.

In the example that is being built for this chapter, a font consisting of four playing card suits is being constructed. This font might consist of only four items, one for each of the playing suits. For example:

```
suitFont.tf_LoChar = 160;
    /* value to use for first character chosen at whim */

suitFont.tf_HiChar = 163;
    /* 160 to 163 range says that there are 4 characters
    * represented in this font */
```

As part of the character data, in addition to defining the included character numbers, you must also define a character representation to be used as the image of a character number requested but not defined in this font. This character is placed at the end of the font definition.

For this example, any character number outside the range of 160-163 inclusive would print this "not in this font" character.

THE CHARACTER DATA

The font structure includes a pointer to the character set data along with descriptions of the how the data is packed into an array. The variables used are defined in *graphics/text.h*; their usage is as follows:

tf_CharData

This is a pointer to the memory location at which the font data begins. This is the bit-packed array of character information.

tf_Modulo

This is the row modulo for the font. The font is organized with the top line of the first character bit adjacent to the top line of the second character and so on.

For example, if the bit-packed character set needs 10 words of 16 bits each to hold the top line of all of the characters in the set, then the value of the modulo will be 20 (bytes). Twenty is the number which must be added to the pointer into the character matrix to go from the first line to the second line of a specific character.

tf_CharLoc

This is a pointer to an array of paired values. The values are the bit offset into the bit-packed character array for this character, and the size of the character in bits. Expressed in C language, this array of values can be expressed as:

```
struct charDef = {  
    WORD charOffset;  
    WORD charBitWidth;  
}
```

In the program definition, the array to which **charLoc** points can be expressed as:

```
struct charDef suitDef[5];  
/* define an array of four sets of character and one "not a  
* character" bit-packed placement and width information */
```

For all proportional fonts, there must be one set of descriptors for each character defined in the character set.

tf_CharSpace

This is a pointer to an array of words of proportional spacing information. This is the width of each character rectangle, in other words, how many bits width are used to contain the edge-to-edge width of this character's bit definition.

For example, a narrow character may still be stored within a wide space (see figure 4-3).

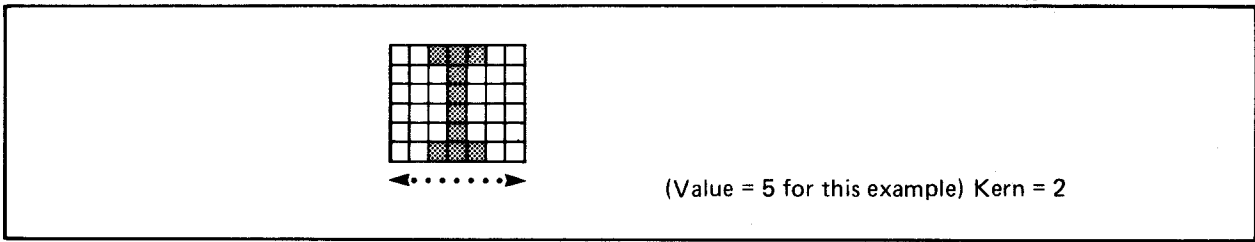


Figure 4-3: CharSpace Figure

If this pointer is null, use the nominal width for each character (**xSize**).

tf_CharKern

This is a pointer to an array of words of character kerning data. Kerning is the offset from the character pointer to the start of the bit data (see figure 4-4). If this pointer is null, kerning is zero.

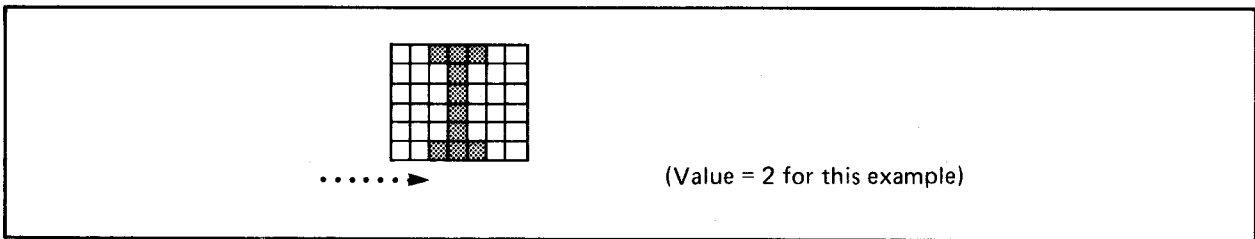


Figure 4-4: CharKern Figure

A COMPLETE SAMPLE FONT

The sample font below pulls together all of the pieces from the above sections. It defines a font whose contents are the four suits from a set of playing cards: clubs, hearts, spades and diamonds.

The suits are defined as proportionally spaced to provide a complete example, even though each suit could as easily have been defined in a 14-wide-by-8-high matrix. There is an open-centered square, which is used if you ask for a character not defined in this font.

```

* A complete sample font. To test this font, the following must be done:
*
* 1. In the AmigaDOS SYS:fonts directory, install a file named
* test.font, containing 264 bytes.
*
* The first two bytes must contain the value hex 0f00, the identifier
* for a font header.
*
* The next word (2 bytes), should contain the value 0001, which is
* the number of FontContents elements. There will be only one
* font in the directory that this font description covers.
*
* Follow this header material with the ASCII value for 'test/8';
* the next 250 bytes should be set to zero. This represents the
* pathname for AmigaDOS to follow from the directory SYS:fonts to
* reach this test font. 'test' is the directory it should go to and
* '8' is the font file itself, as assembled and linked below.
*
* The next two bytes (as one word) contain the font YSize; in this
* case, 0008.
*
* The next byte contains the font Flags, in this case 00.
*
* The last byte contains the font characteristics, in this case hex 60.
* This says it is a disk-based font (bit 1 set) and the font has been
* removed (bit 7 set), saying that the font is not currently resident.
*
* Summary (all in hex) of test.font file:
*
* 0f00 0001 test/8 ..... 0008 00 60
* word word 256-bytes..... word byte byte
*
* 2. Create a directory named 'test' in SYS:fonts.
*
* Copy the file created by assembling and linking the test font
* below into a file named '8' in subdirectory SYS:fonts/test.
*
* Use the font under the Notepad program or any other. It defines ASCII
* characters 'a' 'b' 'c' and 'd' only. All other characters print an
* "unknown character," a rectangle.
*
*----- Included Files -----

```

```

INCLUDE "exec/types.i"
INCLUDE "exec/nodes.i"

```

INCLUDE "libraries/diskfont.i"

MOVEQ #0,D0 ;provide an easy exit in case somebody
;tries to RUN this file instead of loading it.

RTS
DC.L 0 ; ln_Succ
DC.L 0 ; ln_Pred
DC.B NT_FONT ; ln_Type
DC.B 0 ; ln_Pri
DC.L fontName ; ln_Name
DC.W DFH_ID ; FileID
DC.W 1 ; Revision
DC.L 0 ; Segment

fontName:

DS.B MAXFONTNAME ; Name

font:

DC.L 0 ; ln_Succ
DC.L 0 ; ln_Pred
DC.B NT_FONT ; ln_Type
DC.B 0 ; ln_Pri
DC.L fontName ; ln_Name
DC.L 0 ; mn_ReplyPort
DC.W fontEnd-font ; mn_Length
DC.W 8 ; tf_YSize
DC.B 0 ; tf_Style
DC.B FPF_DESIGNED+FPF_PROPORTIONAL ; tf_Flags
DC.W 14 ; tf_XSize
DC.W 6 ; tf_Baseline

* baseline must be no greater than YSize-1, otherwise algorithmically-

* generated style (italic particularly) can corrupt system memory.

DC.W 1 ; tf_BoldSmear
DC.W 0 ; tf_Accessors
DC.B 97 ; tf_LoChar
DC.B 100 ; tf_HiChar
DC.L fontData ; tf_CharData
DC.W 8 ; tf_Modulo, no of bytes to add to
; data pointer to go from one row of
; a character to the next row of it.
DC.L fontLoc ; tf_CharLoc, bit position in the
; font data at which the character
; begins.
DC.L fontSpace ; tf_CharSpace
DC.L fontKern ; tf_CharKern

```

*****
* These are the suits characters that this font data defines. ASCII lower-case
* a,b,c,d. The font descriptor says that there are 4 characters described here.
* The fifth character in the table is the character to be output when there is
* no character in this character set that matches the ASCII value requested.
*
*
* 97          98          99          100         256
* <          x          x          x          x          >
* @@@@ @@@@ @          @          @          @@@@ @@@@@@@@@@@@@@@@
* @@@@@@ @@@@@@ @@@@@@ @@@@ @@@@@@ @@@ @@@
* @@@@@@@@@@@ @@@@@@@@@@@ @@@@@@ @@ @ @@ @@@ @@@ @@@
* @@@@@@@@ @@@@@@@@@@@@@ @@@@@@@@ @@@@@@@@@@@@@@@ @@@ @@@
* @@@@@@ @@@ @ @@@ @@@@@ @@@ @ @@@ @@@ @@@ @@@
* @@@@ @          @          @@@@ @          @@@ @@@
* @          @@@@@ @          @          @@@@@ @@@@@@@@@@@@@@@@
*
*****

```

fontData:

- DC.W \$071C0,\$08040,\$070FF,\$0F000
- DC.W \$0FBE3,\$0E0E0,\$0F8C0,\$03000
- DC.W \$07FCF,\$0F9F3,\$026C0,\$03000
- DC.W \$03F9F,\$0FFFF,\$0FFC0,\$03000
- DC.W \$01F0E,\$0B9F3,\$026C0,\$03000
- DC.W \$00E00,\$080E0,\$020C0,\$03000
- DC.W \$00403,\$0E040,\$0F8FF,\$0F000
- DC.W \$00000,\$00000,\$00000,\$00000
- DC.W \$00000,\$00000,\$00000,\$00000

* font data is bit-packed edge to edge to save space; that's what the
* fontLoc is all about.

fontLoc:

- DC.L \$00000000B,\$0000B000B,\$000160007,\$0001D000B
- DC.L \$00028000C

* Each pair of words specifies how the characters are bit-packed. For
* example, the first character starts at bit position 0000, and is 000B
* (11) bits wide. The second character starts at bit position 000B and
* is 000B bits wide, and so on. Tells font handler how to unpack the
* bits from the array.

fontSpace:

- DC.W 000012,000012,000008,000012,000013

* fontSpace array: Use a space this wide to contain this character
* when it is printed.

```
fontKern:          DC.W  000001,000001,000001,000001,000001
fontEnd:           END
```

Sample Program

The following sample program asks **AvailFonts()** to make a list of the fonts that are available, then opens a separate window and prints a description of the various attributes that can be applied to the fonts, in the font itself. Notice that not all fonts accept all attributes (garnet9 for example, will not underline). If you run this program, note also that not all fonts are as easily readable in the various bold and italicized modes. This rendering is done in a fixed manner by software and the fonts were not necessarily designed to accept it. It is always best to have a font that has been designed with a bold or italic characteristic built in rather than trying to italicize and embolden an existing plain font.

```
/* "whichfont.c" */

#define AFTABLESIZE 2000

#include "exec/types.h"
#include "exec/io.h"
#include "exec/memory.h"

#include "graphics/gfx.h"
#include "hardware/dmabits.h"
#include "hardware/custom.h"
#include "hardware/blit.h"
#include "graphics/gfxmacros.h"
#include "graphics/copper.h"
#include "graphics/view.h"
#include "graphics/gels.h"
#include "graphics/regions.h"
#include "graphics/clip.h"
#include "exec/exec.h"
#include "graphics/text.h"
#include "graphics/gfxbase.h"
#include "devices/keymap.h"
#include "libraries/dos.h"
#include "graphics/text.h"
```

```
#include "libraries/diskfont.h"
#include "intuition/intuition.h"
```

```
struct AvailFonts *af;
struct AvailFontsHeader *afh;
extern int AvailFonts();
```

```
struct TextFont *tf;
struct TextAttr ta;
```

```
ULONG DosBase;
ULONG DiskfontBase;
ULONG IntuitionBase;
ULONG GfxBase;
```

```
struct NewWindow nw = {
    10, 10,          /* starting position (left,top) */
    620,40,         /* width, height */
    -1,-1,         /* detailpen, blockpen */
    0,              /* flags for IDCMP */
    WINDOWDEPTH|WINDOWSIZING|WINDOWDRAG|SIMPLE_REFRESH|
    ACTIVATE|GIMMEZEROZERO, /* window gadget flags */
    0,              /* pointer to 1st user gadget */
    NULL,          /* pointer to user check */
    "Text Font Test", /* title */
    NULL,          /* pointer to window screen */
    NULL,          /* pointer to super bitmap */
    100,45,        /* min width, height */
    640,200,       /* max width, height */
    WBENCHSCREEN};
```

```
struct Window *w;
struct RastPort *rp;
```

```
SHORT text_styles[] = { FS_NORMAL, FSF_UNDERLINED, FSF_ITALIC, FSF_BOLD,
    FSF_ITALIC | FSF_BOLD, FSF_BOLD | FSF_UNDERLINED,
    FSF_ITALIC | FSF_BOLD | FSF_UNDERLINED };
```

```
char *text[] = { " Normal Text", " Underlined", " Italicized", " Bold",
    " Bold Italics", " Bold Underlined",
    " Bold Italic Underlined" };
```

```
char textlength[] = { 12, 11, 11, 5, 13, 16, 23 };
```

```
char *pointsize[] = { " 0", " 1", " 2", " 3", " 4", " 5", " 6", " 7", " 8", " 9",
    "10", "11", "12", "13", "14", "15", "16", "17", "18", "19",
```

```

    "20","21","22","23","24","25","26","27","28","29",
    "30","31"};

char fontname[40];
char dummy[100];    /* provided for string length calculation */
char outst[100];    /* build something to give to Text, see note in the
                    * program body about algorithmically generated styles */

main()
{
    UBYTE fonttypes;
    int j,k,m;
    SHORT aysize;
    SHORT style;
    SHORT sEnd;    /* numerical position of end of string terminator,
                    * and coincidentally the length of the string. */

    if( (DosBase = OpenLibrary("dos.library", 0)) == NULL) exit(-1);
    if((DiskfontBase=OpenLibrary("diskfont.library",0))==NULL) exit(-4);
    if((IntuitionBase=OpenLibrary("intuition.library",0))==NULL) exit(-2);
    if((GfxBase=OpenLibrary("graphics.library",0))==NULL) exit(-3);

    tf=NULL; /* no font currently selected */
    aysize = AFTABLESIZE; /* show how large a buffer is available */
    fonttypes = 0xff; /* show us all font types */

    afh = (struct AvailFontsHeader *) AllocMem(aysize, MEMF_CLEAR);
    if(afh == NULL) exit(-5);

    printf("\nSearching for Fonts\n");
    AvailFonts(afh, aysize, fonttypes);

    af = (struct AvailFonts *) &afh[1]; /* bypass header to get to the
                                          * first of the availfonts */

    for (j = 0; j < afh->afh_NumEntries; j++)
    {
        if((af->af_Attr.ta_Flags & FPF_REMOVED) ||
            (af->af_Attr.ta_Flags & FPF_REVPATH) ||
            ((af->af_Type&AFF_MEMORY)&&
             (af->af_Attr.ta_Flags&FPF_DISKFONT)))
            ; /* do nothing if font is removed, or if font
                * designed to be rendered rt->left (simple
                * example writes left to right) or if font
                * both on disk and in ram, don't list it twice. */
    }
}

```



```

/* AvailFonts performs an AddFont to the system list; if run twice, you
 * get two entries, one of "af_Type 1" saying that the font is memory-
 * resident, and the other of "af_Type 2" saying the font is disk-based.
 * The third part of the if-statement lets you tell them apart if you
 * are scanning the list for unique elements; it says "if it's in
 * memory and it is from disk, then don't list it because you'll find
 * another entry in the table that says it is not in memory, but is on
 * disk." (Another task might have been using the font as well, creating
 * the same effect.)
 */
else
{
    printf("\nFont name found was: %ls",af->af_Attr.ta_Name);
    printf(" and its point size is: %ld",af->af_Attr.ta_YSize);
    /* Style parameter is in af->af_Attr.ta_Style,
     * Flags parameter is in af->af_Attr.ta_Flags.
     */
}
af++;
}
/* now that we've listed the fonts, let's look at them */

w = (struct Window *)OpenWindow(&nw);
rp = w->RPort;

for(m=0; m<2; m++) /* do normal video, then inverse video */
{

af = (struct AvailFonts *)&afh[1]; /* reset value of af to original */
SetAPen(rp,1);

if(m == 0)SetDrMd(rp,JAM1);
else SetDrMd(rp,JAM1+INVERSVID);

/* now print a line that says what font and what style it is */

for (j=0; j < afh->afh_NumEntries; j++)
{
CStringAppend(&fontname[0],af->af_Attr.ta_Name);
    /* copy name into build-name area */
    /* already has ".font" onto end of it */

ta.ta_Name = &fontname[0];
ta.ta_YSize = af->af_Attr.ta_YSize; /* ask for this size */

```

```

ta.ta_Style = af->af_Attr.ta_Style; /* ask for designed style */
ta.ta_Flags = FPF_ROMFONT|FPF_DISKFONT|
    FPF_PROPORTIONAL|FPF_DESIGNED;
    /* accept it from anywhere it exists */
style = ta.ta_Style;

if(!((af->af_Attr.ta_Flags & FPF_REMOVED) ||
    (af->af_Attr.ta_Flags & FPF_REVPATH) ||
    ((af->af_Type&AFF_MEMORY)&&
    (af->af_Attr.ta_Flags&FPF_DISKFONT))))

/* this is an IF-NOT, the reverse of the earlier if-test on
 * these same parameters
 */
{
    tf = (struct TextFont *) OpenDiskFont(&ta);

    if (tf != 0)
    {
        SetFont(w->RPort, tf);
        for(k=0; k<7; k++)
        {
            style = text_styles[k];
            SetSoftStyle(w->RPort,style,255);
            SetRast(rp,0); /* erase any previous text */
            Move(rp,10,20); /* move down a bit from the top */
            sEnd = CStringAppend(&outst[0],af->af_Attr.ta_Name);
            sEnd = sEnd + CStringAppend(&outst[sEnd], " ");
            sEnd = sEnd + CStringAppend(&outst[sEnd],
                pointsize[af->af_Attr.ta_YSize]);
            sEnd = sEnd + CStringAppend(&outst[sEnd], " Points, ");
            CStringAppend(&outst[sEnd],text[k]);
            Text(rp,&outst[0],CStringAppend(&dummy[0],&outst[0]));

/* Have to build the string before sending it out to text IF
 * ALGORITHMICALLY GENERATING THE STYLE since the kerning and
 * spacing tables are based on the vanilla text, and not the
 * algorithmically generated style. If you send characters out
 * individually, it is possible that the enclosing rectangle of
 * a later character will chop off the trailing edge of a
 * preceding character.
 */

```

```

/*****
* This alternate method, when in INVERSVID, exhibits the problem described above.
*
* Text(rp,af->af_Attr.ta_Name,STRLEN(af->af_Attr.ta_Name));
* Text(rp," ",2);
* Text(rp,pointsize[af->af_Attr.ta_YSIZE],2);
* Text(rp," Points, ",9);
*
* Text(rp,text[k],textlength[k]);
*****/

        Delay(40); /* use the DOS time delay function
                   * specifies 60ths of a second */
    }
    CloseFont(tf); /* close the old one */

/* NOTE: Even though you close a font, it doesn't get unloaded from
* memory unless a font with a different name is specified for loading.
* In this case, any font that has been closed (except the topaz set)
* can have its memory area freed, and that font will no longer be
* accessible. If you close a font to go to a different point size, it
* will NOT cause a disk access.
*/

    } /* end of if-tf-ne-0 */
    } /* end of if-(in memory but from disk) */
    af++;
    } /* Do next font now */
} /* end of for-loop, controlled by m */

FreeMem(afh,AFTABLESIZE);
CloseWindow(w);
CloseLibrary(IntuitionBase);
CloseLibrary(DosBase);
CloseLibrary(DiskfontBase);
CloseLibrary(GfxBase);
}

/* copy a string and return the number of characters added to a string.
* Effectively returns the length of the string if not adding anything */

```

```

int CStringAppend(dest, source)
char *dest;
char *source;
{
    int i=0;
    char *s = source;
    char *d = dest;
    while (( i < 79 )&&( *d = *s )) { d++; s++; i++; }
    /* if a NULL found in source, end the copy, but the NULL itself gets
    * copied over to the destination. If no NULL, then 79 characters get
    * copied, then a terminating NULL is added */
    if(i < 79) return(i);
    else { *d = 0; return(i); }
    /* value returned is the position of the terminating NULL to
    * allow other strings to be appended simply using the next
    * append command in sequence */
}

```


PART II

Chapter 5

Audio Device

Introduction

The Amiga has four hardware audio channels—two of the channels produce audio output from the left audio connector and two from the right. These channels can be used in many ways. You can combine a right and a left channel for stereo sound, use a single channel, or play a different sound through each of the four channels.

The audio software is implemented as a standard Amiga input/output device with commands that allocate audio channels and control the sound output.

Some of the audio device commands isolate the programmer from idiosyncrasies of the special-chip hardware. You can also produce sound on the Amiga by directly accessing the hardware registers. For certain types of sound synthesis, this is more CPU-efficient. Some of the audio commands make most sound synthesis easier. Other commands enable your program to co-reside with other programs using the multitasking environment to produce sound at the same time. Programs can co-reside because the audio device handles allocation of audio channels and arbitrates among programs competing for the same resources.

Most personal computers that produce sound have hardware designed for one *specific* synthesis technique. The Amiga uses a very general method of digital sound synthesis that is quite similar to the method used in digital hi-fi components and state-of-the-art keyboard and drum synthesizers, with one significant difference. The Amiga has a tightly-coupled 68000 microprocessor capable of generating and modifying the digital data while the sound is playing. How much of the CPU you can afford to use for sound synthesis depends on your application.

For programs that can afford the memory, playing sampled sounds gives you a simple and very CPU-efficient method of sound synthesis. When a sound is sampled, the amplitude of the waveform that represents a sound is measured (sampled) by an analog-to-digital converter at a fixed interval (period) in time. This results in a table of numbers. When the sound is played back by the Amiga, the table is fed by a DMA channel into one of the four digital-to-analog converters in the custom chips. The digital-to-analog converter converts the samples into voltages that can be played through amplifiers and loudspeakers, reproducing the sound.

On the Amiga you can create sound data in many other ways. For instance, you can use trigonometric functions in your programs to create the more traditional sounds—sine waves, square waves, or triangle waves—by using tables that describe their shapes. Then you can combine these waves for richer sound effects by adding the tables together. Once the data is entered, you can modify it with techniques described in the section called “Audio Functions and Commands.”

For information about the limitations of the audio hardware and suggestions for improving system efficiency and sound quality, refer to the *Amiga Hardware Reference Manual*.

The following works are recommended for information about computer sound generation in general:

- o *Musical Applications of Microprocessors*, by Hal Chamberlain (Hayden, 1980)
- o *Foundations of Computer Music*, by Curtis Roads and John Strawn (Cambridge: MIT Press, 1985)

- o *Digital Audio Signal Processing*, by John Strawn (Los Altos, California: William Kaufmann, Inc., 1985)

Definitions

Terms used in the following discussions may be unfamiliar. Some of the more important terms are defined below.

Amplitude

The height of a waveform, which corresponds to the amount of voltage or current in the electronic circuit.

Amplitude modulation

A means of producing special audio effects by using one channel to alter the amplitude of another.

Buffer

An area of continuous memory, typically used for storing blocks of data.

Channel

One "unit" of the audio device.

Cycle

One repetition of a waveform.

Frequency

The number of times per second a waveform repeats.

Frequency modulation

A means of producing special audio effects by using one channel to affect the period of the waveform produced by another channel.

Period

The time elapsed between the output of successive sound samples, in units of system clock ticks.

Precedence

Priority of the user of a sound channel.

Sample

Byte of audio data, one of the fixed-interval points on the waveform.

Volume

The decibel level of sound coming from an audio channel.

Waveform

Graph that shows a model of how the amplitude of a sound varies over time—usually over one cycle.

Audio Functions and Commands

The first part of this section gives some general information about audio functions and commands. Following the general information there is a brief description of each command. For complete specifications, see the command and function reference section and the header files *devices/audio.i* and *devices/audio.h* in the “Include Files” appendix.

AUDIO AS A DEVICE

The audio device has much in common with the other I/O devices, so general information about device I/O is not repeated here. Before reading further, you should become familiar with the general description of device I/O in the *Amiga ROM Kernel Reference Manual: Exec*.

Audio device commands use an extended **IORequest** block instead of the standard **IORequest** block. When using an audio command, refer to the *devices/audio.i* and *devices/audio.h* files for the extended fields.

SCOPE OF COMMANDS

All audio commands (except for **CMD_WRITE**, **ADCMD_WAITCYCLE**, and **CMD_READ**) can operate on multiple channels. **CMD_WRITE**, **ADCMD_WAITCYCLE**, and **CMD_READ** operate on only one channel. You tell the audio device driver which channels you want a command to act upon by setting the least significant four bits of the **io_unit** field of the **IORequest** block. You specify a 1 in the position of the channel you want to affect and a 0 in all other positions. For instance, you specify 5 (0101) to use channels 0 and 2.

Certain of the audio device commands are actually higher-level functions in that they execute more than one audio device command with a single call. For example, the **OpenDevice()** function, when used for the audio device, can perform an **ADCMD_ALLOCATE** command so that you can start writing data immediately. The **CloseDevice()** function can perform a **ADCMD_FREE** command to relinquish the channel(s) so you can exit immediately after closing the audio device.

ALLOCATION AND ARBITRATION

You request the use of one or more audio channels by performing the `ADCMD_ALLOCATE` command. If possible, `ADCMD_ALLOCATE` obtains the channels for you. When you request a channel, you specify a precedence number from -128 (the lowest precedence) to 127 (the highest). If a channel you want is being used and you have specified a higher precedence than the current user, `ADCMD_ALLOCATE` will “steal” the channel from the other user. Later on, if your precedence is lower than that of another user who is performing an allocation, the channel may be stolen from you. If, after allocating a channel with the appropriate precedence, you raise the precedence to the maximum precedence with the `ADCMD_SETPREC` command, then no other allocation call can steal a channel from you. When you have finished with a channel, you must relinquish it with the `ADCMD_FREE` command to make it available for other users.

Table 5-1 shows suggested precedence values.

Table 5-1: Suggested Precedences for Channel Allocation

Precedence	Type of Sound
127	<i>Unstoppable.</i> Sounds first allocated at lower precedence, then set to this highest level.
90 - 100	<i>Emergencies.</i> Alert, urgent situation that requires immediate action.
80 - 90	<i>Annunciators.</i> Attention, bell (CTRL-G).
75	<i>Speech.</i> Synthesized or recorded speech (narrator.device).
50 - 70	<i>Sonic cues.</i> Sounds that provide information that is not provided by graphics. Only the beginning of each sound (enough to recognize it) should be at this level; the rest should be set to sound effects level.
-50 - 50	<i>Music program.</i> Musical notes in music-oriented program. The higher levels should be used for the attack portions of each note. Notes should separately allocate channels at the start and free them at the end.
-70 - 0	<i>Sound effects.</i> Sounds used in conjunction with graphics. More important sounds should use higher levels.
-100 - -80	<i>Background.</i> Theme music and restartable background sounds.
-128	<i>Silence.</i> Lowest level (freeing the channel completely is preferred).

When you first perform a channel allocation request, the audio device provides you with an “allocation key” that is unique to the granting of your current allocation request. The allocation key is also copied in the **ioa_AllocKey** field of your I/O control block and is used by all audio commands. Later, as you queue output requests to the audio device, the device can compare the allocation key in your request block to the key currently assigned for that channel (or channels). If the channel is stolen from you by another channel user that has a higher precedence, the copy of the key maintained by the audio channel is changed. If you attempt to perform a command on a channel that has been stolen from you, an **AUDIO_NOALLOCATION** error is returned and the bit in the **io_unit** field corresponding to the stolen channel is cleared so you know which channel was stolen.

There is no specific separate “audio resource.” Instead, the audio device, with its allocation key management, arbitrates the use of the physical audio resources.

PERFORMING AUDIO COMMANDS

To perform an audio command, sometimes you must use the system function **BeginIO()** rather than **SendIO()** or **DoIO()**. This is because the latter two functions clear the device-specific bits in the **io_Flags** field of the **IORequest** (bits 4 thru 7). Some of the audio commands use these bits to select options. If you use **SendIO()** or **DoIO()**, the flags will be set to 0 (**FALSE**), which may not be desirable.

COMMAND TYPES

Commands and functions for audio use can be divided into three categories: system functions, allocation/arbitration commands, and hardware control commands. There are also three audio device flags.

The system functions are

- o **OpenDevice()**
- o **CloseDevice()**
- o **BeginIO()**
- o **AbortIO()**

The allocation/arbitration commands are

- o ADCMD_ALLOCATE
- o ADCMD_FREE
- o ADCMD_SETPREC
- o ADCMD_LOCK

The hardware control commands are

- o CMD_WRITE
- o ADCMD_FINISH
- o ADCMD_PERVOL
- o CMD_FLUSH
- o CMD_RESET
- o ADCMD_WAITCYCLE
- o CMD_STOP
- o CMD_START
- o CMD_READ

The following paragraphs describe each function and command.

SYSTEM FUNCTIONS

These are standard Amiga device functions. They are used for communication with the device.

OpenDevice()

The audio device adds to the normal operation of this function. When you open the audio device with a nonzero **ioa_Length** field, **OpenDevice()** will attempt to allocate channels based on allocation mask just as if you had called the ADCMD_ALLOCATE command. This allocation is done with the ADIOF_NOWAIT flag set, so ADCMD_ALLOCATE will return immediately if it fails. If you are opening the device and are not ready to have a channel allocated to you just then, set the **ioa_Length** field to zero.

CloseDevice()

When used with the audio device, **CloseDevice()** performs an `ADCMD_FREE` command on any channels selected by the `io_Unit` field. If you have different allocation keys for the channels you are using, you cannot use this function to close all of them at once. Instead, you will have to issue one `ADCMD_FREE` command for each unique allocation that you are using. After issuing the `ADCMD_FREE` command(s), you can call **CloseDevice()**.

BeginIO()

Audio use of this function differs from normal use only in that it takes a pointer to an **IOAudio** structure as its only argument.

AbortIO()

This function can be used to cancel requests for `ADCMD_ALLOCATE`, `ADCMD_LOCK`, `CMD_WRITE`, or `ADCMD_WAITCYCLE`. When used with the audio device, **AbortIO()** always succeeds.

ALLOCATION/ARBITRATION COMMANDS

These commands allow the audio channels to be shared among different tasks and programs. None of these commands can be called from interrupt code.

ADCMD_ALLOCATE

This command gives access to channels. You perform this command with a pointer to a data array that describes the channels you want to allocate. For example, if you want a pair of stereo channels and you have no preference about which of the left and right channels the system will choose for the allocation, you can pass the command a pointer to an array containing 3, 5, 10, and 12. Channels 0 and 3 output sound on the left side, and channels 1 and 2 on the right side. Table 5-2 shows how this array corresponds to all the possible combinations of a right and a left channel.

Table 5-2: Possible Channel Combinations

Channel 3 left	Channel 2 right	Channel 1 right	Channel 0 left	Decimal Value of Allocation Mask
0	0	1	1	3
0	1	0	1	5
1	0	1	0	10
1	1	0	0	12

How ADCMD_ALLOCATE Operates. The ADCMD_ALLOCATE command tries the first combination, 3, to see if channels 0 and 1 are not being used. If they are available, the 3 is copied into the **io_unit** field and you get an allocation key for these channels. You copy the key into other I/O blocks for the other commands you may want to perform using these channels. When finished with the channels, you perform the ADCMD_FREE command. If channels 0 and 1 are being used, ADCMD_ALLOCATE tries the other combinations in turn. If all the combinations are in use, ADCMD_ALLOCATE checks the precedence number of the users of the channels and finds the combination that requires it to steal the channel or channels of the lowest precedence. If all the combinations require stealing a channel or channels of equal or higher precedence, the I/O request ADCMD_ALLOCATE fails. Precedence is in the **ln_Pri** field of the **io_Message** in the **IORequest** block you pass to ADCMD_ALLOCATE; it has a value from -128 to 127.

The ADIOF_NOWAIT Flag. If you need to produce a sound right now and otherwise you don't want to allocate, set the ADIOF_NOWAIT flag to 1. This will cause the command to return an IOERR_ALLOCFAILED error if it cannot allocate any of the channels. If you are producing a non-urgent sound and you can wait, set the ADIOF_NOWAIT flag to 0. Then, the **IORequest** block returns only when you gets the allocation. If ADIOF_NOWAIT is set to 0, the audio device will continue to retry the allocation request whenever channels are freed until it is successful. If the program decides to cancel the request, **AbortIO()** can be used.

ADCMD_ALLOCATE Examples. The following are some more examples of how to tell ADCMD_ALLOCATE your channel preferences. If you want any channel, but want to try to get a left channel first, use an array containing 1, 8, 2, and 4:

```
0001
1000
0010
0100
```


If you want only a left channel, use 1 and 8 (channels 0 and 3):

0001
1000

For a right channel, use 2 and 4 (channels 1 and 2):

0010
0100

To produce special effects, such as hardware-controlled amplitude and frequency modulation, you may need to allocate channels that can be “attached” to each other. The following allocation map specifies the allowable combinations. (For further information about amplitude and frequency modulation, see the *Amiga Hardware Reference Manual*.)

0011 3
0110 6
1100 12

If you want all the channels, use the following allocation map:

1111 15

If you want to allocate a channel and keep it for a sound that can be interrupted and restarted, allocate it at a certain precedence. If it gets stolen, allocate it again with the `ADIOF_NOWAIT` flag set to 0. When the channel is relinquished, you will get it again.

The Allocation Key. If you want to perform multi-channel commands, all the channels must have the same key since the `IORequest` block has only one allocation key field. The channels must all have that same key even when they were not allocated simultaneously. If you want to use a key you already have, you can pass in that key in the allocation key field and `ADCMD_ALLOCATE` can allocate other channels with that existing key. The `ADCMD_ALLOCATE` command returns a new and unique key only if you pass in a zero in the allocation key field.

ADCMD_FREE

`ADCMD_FREE` is the opposite of `ADCMD_ALLOCATE`. When you perform `ADCMD_FREE` on a channel, it does a `CMD_RESET` command on the hardware and “unlocks” the channel. It also checks to see if there are other pending allocation requests. You do not need to perform `ADCMD_FREE` on channels stolen from you.

ADCMD_SETPREC

This command changes the precedence of an allocated channel. As an example of the use of ADCMD_SETPREC, assume that you are making sound of a chime that takes a long time to decay. It is important that user hears the chime but not so important that he hears it decay all the way. You could lower precedence after the initial attack portion of the sound to let another program steal the channel. You can also set the precedence to maximum (127) if you cannot have the channel(s) stolen from you.

ADCMD_LOCK

The ADCMD_LOCK command performs the “steal verify” function. When a user is attempting to steal a channel or channels, ADCMD_LOCK gives you a chance to clean up before the channel is stolen. You perform a ADCMD_LOCK command right after the ADCMD_ALLOCATE command. ADCMD_LOCK does not return until a higher-priority user attempts to steal the channel(s) or you perform an ADCMD_FREE command. If someone is attempting to steal, you must finish up and ADCMD_FREE the channel as quickly as possible.

ADCMD_LOCK is necessary only if you want to store directly to the hardware registers instead of using the device commands. If your channel is stolen, you are not notified unless the ADCMD_LOCK command is present, and this could cause problems for the user who has stolen the channel and is now using it. ADCMD_LOCK sets a switch that is not cleared until you perform an ADCMD_FREE command on the channel. Canceling an ADCMD_LOCK request with **AbortIO()** will not free the channel.

The following outline describes how ADCMD_LOCK works when a channel is stolen and when it is not stolen.

1. User A allocates a channel.
2. User A locks the channel.

If User B allocates the channel with a higher precedence:

3. User B's ADCMD_ALLOCATE command is suspended (regardless of the setting of the ADIOF_NOWAIT flag).
4. User A's ADCMD_LOCK command is replied to with an error (ADIOERR_CHANNELSTOLEN).
5. User A does whatever is needed to finish up when a channel is stolen.

6. User A frees the channel with `ADCMD_FREE`.
7. User B's `ADCMD_ALLOCATE` command is replied to. Now user B has the channel.

If the channel is not allocated by another user:

3. User A finishes the sound.
4. User A performs the `ADCMD_FREE` command.
5. User A's `ADCMD_LOCK` command is replied.

Never make the freeing of a channel (if the channel is stolen) dependent on allocating another channel. This may cause a deadlock. To keep a channel and never let it be stolen, set precedence to maximum (127). Do not use a lock for this purpose.

HARDWARE CONTROL COMMANDS

The following commands change hardware registers and affect the actual sound output.

CMD_WRITE

This is a single-channel command and is the main command for making sounds. You pass the following to `CMD_WRITE`:

- o A pointer to the waveform to be played (must start on a word boundary and must be in memory accessible by the custom chips, `MEMF_CHIP`)
- o The length of the waveform in bytes (must be an even number)
- o A count of how many times you want to play the waveform

If the count is 0, `CMD_WRITE` will play the waveform from beginning to end, then repeat the waveform continuously until something aborts it.

If you want period and volume to be set at the start of the sound, you set the `WRITE` command's `ADIOF_PERVOL` flag. If you do not do this, the previous volume and period for that channel will be used. This is one of the flags that would be cleared by `DoIO()` and `SendIO()`. The `ioa_WriteMsg` field in the `IORequest` block is an extra message field that can be replied at the start of the `CMD_WRITE`. This second message is used only to tell you when the `CMD_WRITE` command *starts* processing, and it is used only when the `ADIOF_WRITEMESSAGE` flag is set to 1.

If a `CMD_STOP` has been performed, the `CMD_WRITE` requests are queued up.

The `CMD_WRITE` command does not make its own copy of the waveform, so any modification of the waveform before the `CMD_WRITE` command is finished may affect the sound. This is sometimes desirable for special effects.

To splice together two waveforms without clicks or pops, you must send a separate, second `CMD_WRITE` command while the first is still in progress. This technique is used in double-buffering, which is described below.

Double-buffering. By using two waveform buffers and two `CMD_WRITE` requests you can compute a waveform continuously. This is called double-buffering. The following describes how you use double-buffering.

1. Compute a waveform in memory buffer A.
2. Issue `CMD_WRITE` command A with `io_Data` pointing to buffer A.
3. Continue the waveform in memory buffer B.
4. Issue `CMD_WRITE` command B with `io_Data` pointing to Buffer B.
5. Wait for `CMD_WRITE` command A to finish.
6. Continue the waveform in memory buffer A.
7. Issue `CMD_WRITE` command A with `io_Data` pointing to Buffer A.
8. Wait for `CMD_WRITE` command B to finish.
9. Loop back to step 3 until the waveform is finished.
10. At the end, remember to wait until both `CMD_WRITE` command A and `CMD_WRITE` command B are finished.

ADCMD_FINISH

The `ADCMD_FINISH` command aborts (calls `AbortIO()`) the current write request on a channel or channels. This is useful if you have something playing, such as a long buffer or some repetitions of a buffer, and you want to stop it.

`ADCMD_FINISH` has a flag you can set (`ADIOF_SYNC CYCLE`) that allows the waveform to finish the current cycle before aborting it. This is useful for splicing together sounds at zero crossings or some other place in the waveform where the amplitude at the end of one waveform

matches the amplitude at the beginning of the next. Zero crossings are positions within the waveform at which the amplitude is zero. Splicing at zero crossings gives you fewer clicks and pops when the audio channel is turned off or the volume is changed.

ADCMD_PERVOL

ADCMD_PERVOL lets you change the volume and period of a CMD_WRITE that is in progress. The change can take place immediately or you can set the ADIOF_SYNC CYCLE flag to have the change occur at the end of the cycle. This is useful to produce vibratos, glissandos, tremolos, and volume envelopes in music or to change the volume of a sound.

CMD_FLUSH

CMD_FLUSH aborts (calls **AbortIO()**) all CMD_WRITEs and all ADCMD_WAITCYCLEs that are queued up for the channel or channels. It does not abort ADCMD_LOCKs (only ADCMD_FREE clears locks).

CMD_RESET

CMD_RESET restores all the audio hardware registers. It clears the attach bits, restores the audio interrupt vectors if the programmer has changed them, and performs the CMD_FLUSH command to cancel all requests to the channels. CMD_RESET also unstops channels that have had a CMD_STOP performed on them. CMD_RESET does not unlock channels that have been locked by ADCMD_LOCK.

ADCMD_WAITCYCLE

This is a single-channel command. ADCMD_WAITCYCLE is replied to when the current cycle has completed, that is, after the current CMD_WRITE command has reached the end of the current waveform it is playing. If there is no CMD_WRITE in progress, it returns immediately.

CMD_STOP

This command stops the current write cycle immediately. If there are no CMD_WRITEs in progress, it sets a flag so any future CMD_WRITEs are queued up and do not begin processing (playing).

CMD_START

CMD_START undoes the **CMD_STOP** command. Any cycles that were stopped by the **CMD_STOP** command are actually lost because of the impossibility of determining exactly where the DMA ceased. If the **CMD_WRITE** command was playing two cycles and the first one was playing when **CMD_STOP** was issued, the first one is lost and the second one will be played.

This command is also useful when you are playing the same wave form with the same period out of multiple channels. If the channels are stopped, when the **CMD_WRITE** commands are issued, **CMD_START** exactly synchronizes them, avoiding cancellation and distortion. When channels are allocated, they are effectively started by the **CMD_START** command.

CMD_READ

CMD_READ is a single-channel command. Its only function is to return a pointer to the current **CMD_WRITE** command. It enables you to determine which request is being processed.

Example Programs

STEREO SOUND EXAMPLE

This program demonstrates allocating a stereo pair of channels using the allocation/arbitration commands. For simplicity, it uses no hardware control commands and writes directly to the hardware registers. To prevent another task from stealing the channels before writing to the registers, it locks the channels.

```

/*****
*
* Stereo Sound Example
*
* Sam Dicker
* 3 December 1985
* (created: 17 October 1985)
*
*****/

```

```

/* If you are using the Amiga C compiler, turn off stack-checking
* in phase 2, e.g., "lc2 -v filename.q."
*/

```

```

#include "exec/types.h"
#include "exec/memory.h"
#include "hardware/custom.h"
#include "hardware/dmabits.h"
#include "libraries/dos.h"
#include "devices/audio.h"

```

```

/* audio channel assignment */
#define LEFT0B      0
#define RIGHT0B     1
#define RIGHT1B     2
#define LEFT1B      3
#define LEFT0F      1
#define RIGHT0F     2
#define RIGHT1F     4
#define LEFT1F      8

```

```

/* used by example sound */
#define WAVELENGTH  2
#define CLOCK       3579545
#define LEFTFREQ    50.0
#define RIGHTFREQ   50.1
#define MAXVOLUME   64
#define SOUNDPREC   -40

```

```

extern struct MsgPort *CreatePort();
extern struct AudChannel aud[];
extern UWORD dmacon;

```

```

/* four possible stereo pairs */
UBYTE allocationMap[] = {

```

```

LEFT0F | RIGHT0F,
LEFT0F | RIGHT1F,
LEFT1F | RIGHT0F,
LEFT1F | RIGHT1F
};

struct IOAudio *allocIOB = 0; /* used by cleanUp to determine
                               * what needs to be 'cleaned up' */
struct IOAudio *lockIOB = 0;
struct Device *device = 0;
struct MsgPort *port = 0;
BYTE *squareWaveData = 0;

main()
{
    UBYTE channels;
    struct AudChannel *leftRegs, *rightRegs;

    /* allocate I/O blocks from chip public memory and initialize to zero */
    if (((allocIOB = (struct IOAudio *)AllocMem(sizeof(struct IOAudio),
        MEMF_PUBLIC | MEMF_CLEAR)) == 0) ||
        ((lockIOB = (struct IOAudio *)AllocMem(sizeof(struct IOAudio),
        MEMF_PUBLIC | MEMF_CLEAR)) == 0))
        cleanUp("Out of memory");

    /* open the audio device */

    if (OpenDevice(AUDIONAME, 0, allocIOB, 0) != 0)
        cleanUp("Cannot open audio device");
    device = allocIOB->ioa_Request.io_Device;

    /* initialize I/O block for channel allocation */

    allocIOB->ioa_Request.io_Message.mn_Node.ln_Pri = SOUNDPREC;
    if ((port = CreatePort("sound example", 0)) == 0)
        cleanUp("Cannot create message port");
    allocIOB->ioa_Request.io_Message.mn_ReplyPort = port;
    allocIOB->ioa_Request.io_Command = ADCMD_ALLOCATE;

    /* if no channel is available immediately, abandon allocation */
    allocIOB->ioa_Request.io_Flags = ADIOF_NOWAIT;
    allocIOB->ioa_Data = allocationMap;
    allocIOB->ioa_Length = sizeof(allocationMap);

```



```

/* allocate channels now. Alternatively, ADCMD_ALLOCATE could have been
 * preformed when audio was first OpenDevice'd by setting up ioa_Data and
 * ioa_Length before OpenDevice'ing */

BeginIO(allocIOB);
if (WaitIO(allocIOB))
    cleanUp("Channel allocation failed");

/* initialize I/O block for to lock channels */

lockIOB->ioa_Request.io_Message.mn_ReplyPort = port;
lockIOB->ioa_Request.io_Device = device;

/* one lock command to lock both channels */
lockIOB->ioa_Request.io_Unit = allocIOB->ioa_Request.io_Unit;
lockIOB->ioa_Request.io_Command = ADCMD_LOCK;
lockIOB->ioa_AllocKey = allocIOB->ioa_AllocKey;

/* lock the channels */
SendIO(lockIOB);

/* if lock returned there is an error */
if (CheckIO(lockIOB))

    /* the channel must have been stolen */
    cleanUp("Channel stolen");

/* compute the hardware register addresses */

channels = (ULONG)(allocIOB->ioa_Request.io_Unit);
leftRegs = (channels & LEFT0F) ? &aud[LEFT0B] : &aud[LEFT1B];
rightRegs = (channels & RIGHT0F) ? &aud[RIGHT0B] : &aud[RIGHT1B];

/* allocate waveform memory from chip-addressable ram. AllocMem always
 * allocates memory on a word boundary which is necessary for audio
 * waveform data */

if ((squareWaveData = (BYTE *)AllocMem(WAVELENGTH, MEMF_CHIP)) == 0)
    cleanUp("Out of memory");

/* a two cycle square wave (how complex!) */

squareWaveData[0] = 127;
squareWaveData[1] = -127;

```

```

/* these registers are described in detail in the Amiga Hardware Manual */

/* write-only hardware registers must be loaded separately.
 * <reg1> = <reg2> = <data> may not work with some compilers */
leftRegs->ac_ptr = (UWORD *)squareWaveData;
rightRegs->ac_ptr = (UWORD *)squareWaveData;
leftRegs->ac_len = WAVELENGTH / 2;
rightRegs->ac_len = WAVELENGTH / 2;

/* a slightly different frequency is used in each channel to make the
 * sound a bit more interesting */

leftRegs->ac_per = CLOCK / LEFTFREQ / WAVELENGTH;
rightRegs->ac_per = CLOCK / RIGHTFREQ / WAVELENGTH;

leftRegs->ac_vol = MAXVOLUME;
rightRegs->ac_vol = MAXVOLUME;
dmacon = DMAF_SETCLR | channels << DMAB_AUD0;

/* play sound until the user press CTRL-C or lock is replied*/

puts("Press CTRL-C to stop");
putchar(0);
while(Wait(SIGBREAKF_CTRL_C | 1 << port->mp_SigBit) != SIGBREAKF_CTRL_C)

    /* each time the port signals, check if lock is replied
     * (a signal is not guaranteed to be valid) */

    if (CheckIO(lockIOB)) {
        puts("Channel stolen");
        break;
    }

/* free any allocated audio channels. In this instance explicitly
 * performing the ADCMD_FREE command is unnecessary. CloseDevice'ing
 * with allocIOB performs it and frees the channels automatically */

allocIOB->ioa_Request.io_Command = ADCMD_FREE;
DoIO(allocIOB);

/* free up resources and exit */
cleanUp("");
}

```

```

/* print an error message and free allocated resources */

cleanUp(message)
TEXT *message;
{
    puts(message);
    if (squareWaveData != 0)
        FreeMem(squareWaveData, WAVELENGTH);
    if (port != 0)
        DeletePort(port);
    if (device != 0)
        CloseDevice(allocIOB);
    if (lockIOB != 0)
        FreeMem(lockIOB, sizeof(struct IOAudio));
    if (allocIOB != 0)
        FreeMem(allocIOB, sizeof(struct IOAudio));
    exit();
}

```

DOUBLE-BUFFERED SOUND SYNTHESIS EXAMPLE

This program demonstrates double-buffered writing to an audio channel using the hardware control commands. This technique can be used to synthesize sound in “real-time.” This program uses the mouse as a simple input device; to keep the example simple, the program directly reads the mouse register.

Real-time synthesis code should always be written in the fastest assembly language possible (unlike this example) and should try to precompute as much data as possible. In this example, a sine wave look-up table is precomputed. Then, while the sound is being played, the table is scanned at a rate dependent on a variable (frequency) and the scanned values are copied into temporary buffers. This frequency variable is modified by mouse movement, effectively making the mouse a pitch control. In a “real” program, because pitch is the only parameter being controlled, it would be much more efficient to modify the “period” and play one fixed sine-wave waveform buffer (or one waveform for each octave).

Two temporary buffers are used. One must be computed and sent to the audio device before the other one has finished playing. Otherwise, the audio device turns off the sound, making a pop. This program runs in software interrupts to make sure that it gets adequate processor time to avoid this problem.

```

/*****
 *
 * Double-Buffered Sound Synthesis Example
 *
 * Sam Dicker
 * 3 December 1985 (created: 8 October 1985)
 *
 *****/

/* If you are using the Amiga C compiler, turn off stack-checking
 * in phase 2, e.g., "lc2 -v filename.q."
 */
#include "exec/types.h"
#include "exec/memory.h"
#include "exec/interrupts.h"
#include "exec/errors.h"
#include "hardware/custom.h"
#include "libraries/dos.h"
#include "devices/audio.h"

#define BUFFERSIZE      250
#define SINETABLEPOWER2 10
#define SINETABLESIZE   (1 << SINETABLEPOWER2)
#define SINETABLESTEP   (2 * 3.141593 / SINETABLESIZE)

/* mouse register addresses */
#define XMOUSEREG      (*((BYTE *)&joy0dat + 1))
#define YMOUSEREG      (-(*(BYTE *)&joy0dat))

extern struct MsgPort *CreatePort();
extern struct Library *OpenLibrary();
extern struct Task *FindTask();
extern UWORD joy0dat;

/* channel allocation map */
UBYTE allocationMap[] = { 1, 8, 2, 4 };

struct Library *MathBase = 0; /* used by cleanUp to determine
 * what needs to be 'cleaned up' */
struct MsgPort *allocPort = 0;
struct IOAudio *allocIOB = 0;
struct Device *device = 0;
struct Interrupt *interrupt = 0;
struct MsgPort *soundPort = 0;
BYTE *buffer[2] = { 0 };

```

```

struct IOAudio *soundIOB[2] = { 0 };

int newBuffer();
UBYTE sineTable[SINETABLESIZE];
ULONG angle = 0;
ULONG frequency = 0x2000000;
BYTE lastYMouse;

main()
{
    int i;
    FLOAT sine = 0.0;
    FLOAT cosine = 1.0;

    /* open the math library */

    if ((MathBase = OpenLibrary("mathfp.library", 0)) == 0)
        cleanUp("Cannot open math library");

    /* generate the sine lookup table */

    for (i = 0; i < SINETABLESIZE; ++i) {

        /* generate table values between -128 and 127 */
        sineTable[i] = 127 * sine + 0.5;

        /* compute the next point in the table. The table could have been
         * computed by calling the 'sin' function for each point, but this
         * method is a little faster where great accuracy is not required */
        sine += SINETABLESTEP * (cosine -= SINETABLESTEP * sine);
    }

    /* read the starting mouse count */
    lastYMouse = YMOUSEREG;

    /* initialize I/O block to allocate a channel when the audio device is OpenDevice'd */

    if ((allocPort = CreatePort("sound example", 0)) == 0)
        cleanUp("Cannot create reply port");
    if ((allocIOB = (struct IOAudio *)AllocMem(sizeof(struct IOAudio),
        MEMF_PUBLIC | MEMF_CLEAR)) == 0)
        cleanUp("Out of memory");

    /* allocation precedence */
    allocIOB->ioa_Request.io_Message.mn_Node.ln_Pri = -40;

```

```

allocIOB->ioa_Request.io_Message.mn_ReplyPort = allocPort;

/* allocate from any channel */
allocIOB->ioa_Data = allocationMap;
allocIOB->ioa_Length = sizeof(allocationMap);

/* open the audio device with channel allocation and check for errors */

switch (OpenDevice(AUDIONAME, 0, allocIOB, 0)) {
case IOERR_OPENFAIL:
    cleanUp("Cannot open audio device");
case ADIOERR_ALLOCFAILED:
    cleanUp("Cannot allocate audio channel");
}
device = allocIOB->ioa_Request.io_Device;

/* initialize the software interrupt structure */

if ((interrupt = (struct Interrupt *)AllocMem(sizeof(struct Interrupt),
    MEMF_CLEAR | MEMF_PUBLIC)) == 0)
    cleanUp("Out of memory");
interrupt->is_Code = (VOID (*)())newBuffer;

/* initialize the reply port for CMD_WRITE's to generate software interrupts */

if ((soundPort = (struct MsgPort *)AllocMem(sizeof(struct MsgPort),
    MEMF_CLEAR | MEMF_PUBLIC)) == 0)
    cleanUp("Out of memory");
soundPort->mp_Flags = PA_SOFTINT;
soundPort->mp_SigTask = (struct Task *)interrupt;
soundPort->mp_Node.ln_Type = NT_MSGPORT;
NewList(&soundPort->mp_MsgList);

/* initialize both I/O blocks for the CMD_WRITES */

for (i = 0; i < 2; ++i) {

    /* allocate waveform memory from chip addressable ram. AllocMem
    * always allocates memory on a word boundary which is necessary
    * for audio waveform data */
    if ((buffer[i] = (BYTE *)AllocMem(BUFFERSIZE, MEMF_CHIP))
        == 0)
        cleanUp("Out of memory");

    if ((soundIOB[i] = (struct IOAudio *)AllocMem(sizeof(struct IOAudio),

```

```

MEMF_PUBLIC | MEMF_CLEAR)) == 0)
    cleanUp("Out of memory");
soundIOB[i]->ioa_Request.io_Message.mn_ReplyPort = soundPort;
soundIOB[i]->ioa_Request.io_Device = device;
soundIOB[i]->ioa_Request.io_Unit = allocIOB->ioa_Request.io_Unit;
soundIOB[i]->ioa_Request.io_Command = CMD_WRITE;

/* load the volume and period registers */
soundIOB[i]->ioa_Request.io_Flags = ADIOF_PERVOL;

soundIOB[i]->ioa_AllocKey = allocIOB->ioa_AllocKey;
soundIOB[i]->ioa_Data = buffer[i];
soundIOB[i]->ioa_Length = BUFFERSIZE;

/* some arbitrary period and volume */

soundIOB[i]->ioa_Period = 200;
soundIOB[i]->ioa_Volume = 64;

/* play one cycle of each buffer, then reply */
soundIOB[i]->ioa_Cycles = 1;

/* this really "primes the pump" by causing the reply port
 * to generate a software interrupt and write the first buffers */
ReplyMsg(soundIOB[i]);
}

/* wait for CTRL-C to stop the program */

puts("Press CTRL-C to stop");
putchar(0);
Wait(SIGBREAKF_CTRL_C);

/* free up resources and exit */
cleanUp("");
}

/* print an error message and free allocated resources */

cleanUp(message)
TEXT *message;
{
    int i;

    puts(message);
}

```

```

if (device != 0)

    /* CloseDevice'ing with 'allocIOB' preforms an ADCMD_FREE on any
    * channel allocated with 'allocIOB's ioa_AllocKey. ADCMD_FREE
    * performs a CMD_RESET, which performs a CMD_FLUSH, which AbortIO's
    * any CMD_WRITES to those channels */
    CloseDevice(allocIOB);

for (i = 0; i < 2; ++i) {
    if (soundIOB[i])
        FreeMem(soundIOB[i], sizeof(struct IOAudio));
    if (buffer[i])
        FreeMem(buffer[i], BUFFERSIZE);
}
if (soundPort)
    FreeMem(soundPort, sizeof(struct MsgPort));
if (interrupt)
    FreeMem(interrupt, sizeof(struct Interrupt));
if (allocIOB)
    FreeMem(allocIOB, sizeof(struct IOAudio));
if (allocPort)
    DeletePort(allocPort, sizeof(struct MsgPort));
if (MathBase)
    CloseLibrary(MathBase);
exit();
}

/* software interrupt server code */

newBuffer()
{
    int i;
    struct IOAudio *ioa;
    BYTE *buffer;
    BYTE mouseChange, curYMouse;
    ULONG newFreq;

    /* get I/O block from reply port */
    ioa = (struct IOAudio *)GetMsg(soundPort);

    /* check if there really was an I/O block on the port and if there are no
    * errors. An error would indicate either the channel was aborted from
    * being stolen (IOERR_ABORTED), it was stolen before the write was
    * performed and had the wrong allocation key (ADIOF_NOALLOCATION), or it
    * was aborted by being CloseDevice'd. In any case, if there is an error do

```



```

* not send the next write. The program will just wait around silently */
if (ioa && ioa->ioa_Request.io_Error == 0) {
    /* determine how far the mouse has moved */

    curYMouse = YMOUSEREG;
    mouseChange = curYMouse - lastYMouse;
    lastYMouse = curYMouse;

    /* modify the frequency proportionally */
    newFreq = frequency + mouseChange * (frequency >> 6);

    /* limit the frequency range */
    if (newFreq > 0x800000 && newFreq < 0x40000000)
        frequency = newFreq;

    /* scan the table and copy each new sample into the audio waveform buffer */
    for (i = 0, buffer = ioa->ioa_Data; i < BUFFERSIZE; ++i)
        *buffer++ = sineTable[(angle += frequency) >>
            (32 - SINETABLEPOWER2)];

    /* send the write I/O block */
    BeginIO(ioa);
}
}

```

Chapter 6

Timer Device

Introduction

The Amiga timer device provides a general time-delay capability. It can signal you when *at least* a certain amount of time has passed. Because the Amiga is a multitasking system, the timer device cannot guarantee that exactly the specified amount of time has elapsed.

To use a timer device you open up a channel of communication to the device and send the device a message saying how much time should elapse. At the end of that time, the device returns a message to you stating that the time has elapsed.

Timer Device Units

There are two units in the timer device. One uses the vertical blank interrupt for its “tick” and is called `UNIT_VBLANK`. The other uses a programmable timer in the 8520 CIA chip and is called `UNIT_MICROHZ`. These are the names you use when calling `OpenDevice()`. The examples at the end of the chapter demonstrate how you call `OpenDevice()`.

The `VBLANK` timer unit is very stable and has a precision comparable to the vertical blanking time, that is, ± 16.67 milliseconds. When you make a timing request, such as “signal me in 21 seconds,” the reply will come in $21 \pm .017$ seconds. This timer has very low overhead and should be used for all long duration requests.

The `MICROHZ` timer unit uses the built-in precision hardware timers to create the timing interval you request. It accepts the same type of command—“signal me in so many seconds and microseconds.” The microhertz timer has the advantage of greater resolution than the vertical blank timer, but it has less accuracy over comparable periods of time. The microhertz timer also has much more system overhead. It is primarily useful for short burst timing for which critical accuracy is not required.

Specifying the Time Request

Both timer units have identical external interfaces. Time is specified via a `timeval` structure.

```
struct timeval {
    ULONG tv_secs;
    ULONG tv_micro;
};
```

The time specified is measured from the time the request is posted. For example, you must post a timer request for 30 minutes, rather than for a specific time such as 10:30 p.m. The `micro` field is the number of microseconds in the request. Logically, seconds and microseconds are concatenated by the driver. The number of microseconds must be “normalized;” it should be a value less than one million.

The primary means of specifying a requested time is via a `timeRequest` structure. A time request consists of an `IORequest` structure followed by a `timeval` structure, as shown below.

```

struct timeRequest {
    struct IORequest tr_node;
    struct timeval tr_time;
};

```

Note that the timer driver does not use a “standard extension” **IORequest** block. It only uses the base **IORequest** structure. When the specified amount of time has elapsed, the driver will send the **IORequest** back via **ReplyMsg()** (the same as all other drivers). This means that you must fill in the **ReplyPort** pointer of the **IORequest** structure if you wish to be signaled.

When you submit a timer request, the driver destroys the values you have provided in the **timeval** structure. This means that you must reinitialize the time specification before reposting the **IORequest**.

Multiple requests may be posted to the timer driver. For example, you can make three time requests in a row to the timer, specifying:

```

Signal me in 20 seconds (request 1)
Signal me in 30 seconds (request 2)
Signal me in 10 seconds (request 3)

```

As the timer queues these requests, it changes the time values and sorts the timer requests to service each request at the requested interval, resulting effectively in the following order:

```

(request 3) in now+10 seconds
(request 1) 10 seconds after request 3 is satisfied
(request 2) 10 seconds after request 1 is satisfied

```

A sample timer program is given at the end of this chapter.

Opening a Timer Device

To gain access to a timer unit, you must first open that unit. This is done by using the system command **OpenDevice()**. A typical C-language call is shown below:

```

struct timereq timer_request_block
error = OpenDevice(TIMERNAME,unit_number,timer_request_block,0);

```

The parameters shown above are as follows:

TIMERNAME

This is a define for the null-terminated string, currently "timer.device."

unit_number

This indicates which timer unit you wish to use, either `UNIT_VBLANK` or `UNIT_MICROHZ` as defined in "Timer Device Units" above.

timer_request_block

This is the address of an `IORequest` data structure that will be used later to communicate with the device. The `OpenDevice()` command will fill in the unit and device fields of this data structure.

Adding a Time Request

You add a timer request to the device by passing a correctly initialized I/O request to the timer. The code fragment below demonstrates a sample request:

```
set_timer(seconds,microseconds)
ULONG seconds, microseconds;
{
    timermsg->io_Command = TR_ADDREQUEST;
    timermsg->tr_time.tv_secs = seconds;
    timermsg->tr_time.tv_micro = microseconds;
    DoIO(timermsg);
}
```

Note: Using `DoIO()` here puts your task to sleep until the time request has been satisfied (see the sample program at the end of the chapter).

If you wish to send out multiple time requests, you have to create multiple request blocks (referred here as "timermsg") and then use `SendIO()` to transmit each to the timer.

Closing a Timer

After you have finished using a timer device, you should close it:

`CloseDevice(timermsg);`

Additional Timer Functions and Commands

There are two additional timer commands (accessed as standard device commands, using an `IOResult` block as shown above) and three additional functions (accessed as though they were library functions).

The additional timer commands are as follows:

- o `TR_GETSYSTIME` — get the system time
- o `TR_SETSYSTIME` — set the system time

The additional timer library-like functions are:

- o `SubTime(Dest, Source)` — subtract one time request from another
- o `AddTime(Dest, Source)` — add one time request to another
- o `result = CmpTime(Dest, Source)` — compare the time in two time requests

SYSTEM TIME

The “system timer” is unrelated to the system time as it appears in the `DateStamp` command of AmigaDOS. It is provided simply for the convenience of the developer and is utilized by Intuition.

The command `TR_SETSYSTIME` sets the system’s idea of what time it is. The system starts out at time “zero” so it is safe to set it forward to the “real” time. However, care should be taken when setting the time backwards. System time is specified as being monotonically increasing.

The time is incremented by a special power supply signal that occurs at the external line frequency. This signal is very stable over time, but it can vary by several percent over short periods of time. System time is stable to within a few seconds a day. In addition, system time is changed every time someone asks what time it is using `TR_GETSYSTIME`. This way the return value of the system time is unique and unrepeating. This allows system time to be used as a unique identifier.

Note: The timer device sets system time to zero at boot time. AmigaDOS will set the system time when it reads in the boot disk, if it has not already been set by someone else (more exactly, if the time is less than 86,400 seconds [one day]). AmigaDOS sets the time to the last modification time of the boot disk. The time device does not interpret system time to any physical value. AmigaDOS treats system time relative to midnight, 1 January 1978.

Here is a program that can be used to inquire the system time. Instead of using the Exec support function `CreateStdIO()` for the request block, the block is initialized “correctly” for use as a `timeval` request block. The command is executed by the timer device and, on return, the caller can find the data in his request block.

```
/* getsystime.c - get system time */

#include "exec/types.h"
#include "exec/lists.h"
#include "exec/nodes.h"
#include "exec/ports.h"
#include "exec/io.h"
#include "exec/devices.h"
#include "devices/timer.h"

#define msgblock tr.tr_node.io_Message
struct timerequest tr;

main()
{
    int error;
    error = OpenDevice(TIMERNAME,UNIT_MICROHZ,&tr,0);
    msgblock.mn_Node.ln_Type = NT_MESSAGE;
    msgblock.mn_Node.ln_Pri = 0;
    msgblock.mn_Node.ln_Name = NULL;
    msgblock.mn_ReplyPort = NULL;

    tr.tr_node.io_Command = TR_GETSYSTIME;
    DoIO(&tr);

    printf("\nSystem Time is:\n");
    printf ("Seconds\n");
    printf ("%10ld\n",tr.tr_time.tv_secs, tr.tr_time.tv_micro);
    CloseDevice(&tr);
}

/* end of main */
```

USING THE TIME ARITHMETIC ROUTINES

As indicated above, the time arithmetic routines are accessed in the timer device structure as though it were a routine library. To use them, you create an **IORequest** block and open the timer. In the **IORequest** block is a pointer to the device's base address. This address is needed to access each routine as an offset—for example, **_LVOAddTime**, **_LVOSubTime**, **_LVOCmpTime**—from that base address. (See the “Device Summaries” appendix for these commands.)

There are C-language interface routines in *amiga.lib* that perform this interface task for you. They are accessed through a variable called **TimerBase**. You prepare this variable by the following method (this is only a partial example):

```
struct timeval time1, time2, time3;
SHORT result;

struct Device *TimerBase;    /* declare the interface variable */

TimerBase = timermsg->Device;

/* now that TimerBase is initialized, it is permissible to call
 * the time-comparison or time-arithmetic routines */

time1.tv_secs = 3; time1.tv_micro = 0;           /* 3.0 seconds */
time2.tv_secs = 2; time2.tv_micro = 500000;    /* 2.5 seconds */
time3.tv_secs = 1; time2.tv_micro = 900000;    /* 1.9 seconds */

/* result of this example is +1 ... first parameter has
 * greater time value than second parameter
 */
result = CmpTime( &time1, &time2 );

/* add to time1 the values in time2 */
AddTime( &time1, &time2);
/* subtract values in time3 from the value currently in time1.
 * Results in time1. */
SubTime( &time1, &time3);
```


WHY USE TIME ARITHMETIC?

As mentioned earlier in this section, because of the multitasking capability of the Amiga, the timer device can provide timings that are at least as long as the specified amount of time. If you need more precision than this, using the system timer along with the time arithmetic routines can at least, in the long run, let you synchronize your software with this precision timer after a selected period of time.

Say, for example, that you select timer intervals so that you get 161 signals within each 3-minute span. Therefore, the **timeval** you would have selected would be 180/161, which comes out to 1 second and 118,012 microseconds per interval. Considering the time it takes to set up a call to **set_timer** and delays due to task-switching (especially if the system is very busy) it is possible that after 161 timing intervals, you may be somewhat beyond the 3-minute time. Here is a method you can use to keep in sync with system time:

1. Begin.
2. Read system time; save it.
3. Perform your loop however many times in your selected interval.
4. Read system time again, and compare it to the old value you saved. (For this example, it will be more or less than 3 minutes as a total time elapsed.)
5. Calculate a new value for the time interval (**timeval**); that is, one that (if precise) would put you exactly in sync with system time the next time around. **Timeval** will be a lower value if the loops took too long, and a higher value if the loops didn't take long enough.
6. Repeat the cycle.

Over the long run, then, your average number of operations within a specified period of time can become precisely what you have designed.

Sample Timer Program

Here is an example program showing how to use a timer device.

```

/* Simple Timer Example Program:
 *
 * Includes dynamic allocation of data structures needed to communicate
 * with the timer device as well as the actual device I/O
 */

#include "exec/types.h"
#include "exec/nodes.h"
#include "exec/lists.h"
#include "exec/memory.h"
#include "exec/interrupts.h"
#include "exec/ports.h"
#include "exec/libraries.h"
#include "exec/tasks.h"
#include "exec/io.h"
#include "exec/devices.h"
#include "devices/timer.h"

APTR TimerBase;          /* to get at the time comparison functions */

/* manifest constants -- "never will change" */
#define SECSPERMIN (60)
#define SECSPERHOUR (60*60)
#define SECSPERDAY (60*60*24)

extern struct timerequest *CreateTimer();

main()
{
    /* save what system thinks is the time.... we'll advance it temporarily */
    LONG seconds;
    struct timerequest *tr;
    struct timeval oldtimeval;
    struct timeval mytimeval;
    struct timeval currentval;

    printf("Timer test0);

    /* sleep for two seconds */
    currentval.tv_secs = 2;
    currentval.tv_micro = 0;
    TimeDelay( &currentval, UNIT_VBLANK );
    printf( "After 2 seconds delay0 );

```

```

/* sleep for four seconds */
currentval.tv_secs = 4;
currentval.tv_micro = 0;
TimeDelay( &currentval, UNIT_VBLANK );
printf( "After 4 seconds delay0 );

/* sleep for 500,000 micro-seconds = 1/2 second */
currentval.tv_secs = 0;
currentval.tv_micro = 500000;
TimeDelay( &currentval, UNIT_MICROHZ );
printf( "After 1/2 second delay0 );

printf( "0 );

(void) Execute( "date", 0, 0 );

printf( "0 );

GetSysTime( &oldtimeval );
printf( "Current system time is %ld current seconds0,
        oldtimeval.tv_secs );

printf("Setting a new system time0);

seconds = 1000 * SECSPERDAY + oldtimeval.tv_secs;

SetNewTime( seconds );
/* (if user executes the AmigaDOS DATE command now, he will
 * see that the time has advanced something over 1000 days */

printf( "0 );
(void) Execute( "date", 0, 0 );

printf( "0 );

/* added the microseconds part to show that time keeps
 * increasing even though you ask many times in a row */
GetSysTime( &mytimeval );
printf( "Original system time is %ld.%06ld0,
        mytimeval.tv_secs, mytimeval.tv_micro );

GetSysTime( &mytimeval );
printf( "First system time is %ld.%06ld0,
        mytimeval.tv_secs, mytimeval.tv_micro );

```

```

    GetSysTime( &mytimeval );
    printf( "Second system time is %ld.%06ld0,
           mytimeval.tv_secs, mytimeval.tv_micro );

    printf( "Resetting to former time0 );
    SetNewTime( oldtimeval.tv_secs );

    GetSysTime( &mytimeval );
    printf( "Current system time is %ld.%06ld0,
           mytimeval.tv_secs, mytimeval.tv_micro );

    /* just shows how to set up for using the timer functions, does not
     * demonstrate * the functions themselves. (TimerBase must have a
     * legal value before AddTime, SubTime or CmpTime are performed. */
    tr = CreateTimer( UNIT_MICROHZ );
    TimerBase = (APTR)tr->tr_node.io_Device;

    /* and how to clean up afterwards */
    TimerBase = (APTR)(-1);
    DeleteTimer( tr );
}

extern struct MsgPort *CreatePort();
extern struct IORequest *CreateExtIO();

struct timerequest *
CreateTimer( unit )
ULONG unit;
{
    /* return a pointer to a time request. If any problem, return NULL */

    int error;

    struct MsgPort *timerport;
    struct timerequest *timermsg;

    timerport = CreatePort( 0, 0 );
    if( timerport == NULL )
    {
        return( NULL );
    }

    timermsg = (struct timerequest *)
        CreateExtIO( timerport, sizeof( struct timerequest ) );

```

```

    if( timermsg == NULL ) {
        return( NULL );
    }

    error = OpenDevice( TIMERNAME, unit, timermsg, 0 );
    if( error != 0 )
    {
        DeleteTimer( timermsg );
        return( NULL );
    }

    return( timermsg );
}

/* more precise timer than AmigaDOS Delay() */
TimeDelay( tv, unit )
struct timeval *tv;
int unit;
{
    struct timerequest *tr;

    /* get a pointer to an initialized timer request block */
    tr = CreateTimer( unit );

    /* any nonzero return says timedelay routine didn't work. */
    if( tr == NULL ) return( -1 );

    WaitForTimer( tr, tv );

    /* deallocate temporary structures */
    DeleteTimer( tr );
    return( 0 );
}

int
WaitForTimer( tr, tv )
struct timerequest *tr;
struct timeval *tv;
{
    tr->tr_node.io_Command = TR_ADDREQUEST; /* add a new timer request */

    /* structure assignment */
    tr->tr_time = *tv;

    /* post request to the timer -- will go to sleep till done */

```

```

        DoIO( tr );
    }

int
SetNewTime( secs )
LONG secs;    /* seconds since 1 Jan 78 */
{
    struct timerequest *tr;

    tr = CreateTimer( UNIT_MICROHZ );

    /* non zero return says error */
    if( tr == 0 ) return( -1 );

    tr->tr_node.io_Command = TR_SETSYSTIME;
    tr->tr_time.tv_secs = secs;
    tr->tr_time.tv_micro = 0;
    DoIO( tr );

    DeleteTimer(tr);
    return(0);
}

int
GetSysTime(tv)
struct timeval *tv;
{
    struct timerequest *tr;

    tr = CreateTimer( UNIT_MICROHZ );

    /* non zero return says error */
    if( tr == 0 ) return( -1 );

    tr->tr_node.io_Command = TR_GETSYSTIME;
    DoIO( tr );

    /* structure assignment */
    *tv = tr->tr_time;

    DeleteTimer( tr );
    return( 0 );
}

```

```

int
DeleteTimer( tr )
struct timerequest *tr;
{
    struct MsgPort *tp;

    if( tr != 0 )
    {
        tp = tr->tr_node.io_Message.mn_ReplyPort;
        if(tp != 0) {
            DeletePort(tp);
        }

        CloseDevice( tr );
        DeleteExtIO( tr, sizeof(struct timerequest) );
    }
}

```

Chapter 7

Trackdisk Device

Introduction

The Amiga trackdisk device directly drives the disk, controls the disk motors, reads raw data from the tracks, and writes raw data to the tracks. Normally, you use the AmigaDOS functions to write or read data from the disk. The trackdisk driver is the lowest-level software access to the disk data and is used by AmigaDOS to get its job done. The trackdisk device supports the usual commands such as `CMD_WRITE` and `CMD_READ`. In addition, it supports an extended form of these commands to allow additional control over the disk driver.

The trackdisk device can queue up command sequences so that your task can do something else while it is waiting for a particular disk activity to occur. If several sequenced write commands are queued to a disk, a task assumes that all such writes are going to the same disk. The trackdisk driver itself can stop a command sequence if it senses that the disk has been changed, returning all subsequent **IORequest** blocks to the caller with an error (“disk changed”).

When the trackdisk device is requested to provide status information for commands such as **TD_REMOVE** or **TD_CHANGENUM**, the value is returned in the **io_Actual** field of the **IORequest**.

The Amiga Floppy Disk

The Amiga floppy disk consists of **NUMHEADS** (2) heads, **NUMCYLS** (80) cylinders, and **NUMSECS** (11) sectors per cylinder. Each sector has **TD_SECTOR** (512) usable data bytes plus **TD_LABELSIZE** (16) of sector label area. This gives useful space of 880K bytes plus 28K bytes of label area per floppy disk.

Although the disk is logically divided up into sectors, all I/O to the disk is implemented as an entire track. This allows access to the drive with no interleaving and increases the useful storage capacity by about 20 percent. Normally, a read of a sector will only have to copy the data from the track buffer. If the track buffer contains another track’s data, then the buffer will first be written back to the disk (if it is “dirty”) and the new track will be read in. All track boundaries are transparent to the user. The driver ensures that the correct track is brought into memory.

The performance of the disk is greatly enhanced if you make effective use of the track buffer. The performance of sequential reads will be up to an order of magnitude greater than reads scattered across the disk.

The disk driver uses the blitter to encode and decode the data to and from the track buffer. Because the blitter can access only chip memory (memory that is accessible to the special-purpose chips and within the lowest 512K bytes of the system, known as **MEMF_CHIP** to the memory allocator **AllocMem()**), all buffers submitted to the disk must be in chip memory. In addition, only full-sector writes on sector boundaries are supported. Note also that the user’s buffer must be word-aligned.

The disk driver is based upon a standard driver structure. It has the following restrictions:

- o All reads and writes must use an **io_Length** that is an integer multiple of **TD_SECTOR** bytes (the sector size in bytes).

- o The offset field must be an integer multiple of TD_SECTOR.
- o The data pointer must be word-aligned.
- o The data pointer must be in MEMF_CHIP memory. This is because the disk driver uses the blitter to fill the data buffer.
- o Only the 3 1/2-inch disk format is supported by the trackdisk driver. The 5 1/4-inch format is supported by the IBM PC emulation software.

Trackdisk Driver Commands

The trackdisk driver allows the following system interface functions and commands. In addition to the usual device commands, the trackdisk driver has a set of extended commands.

The system interface functions are

OpenDevice()	Obtain exclusive use of a particular disk unit
CloseDevice()	Release the unit to another task
Expunge()	Remove the device from the device list
BeginIO()	Dispatch a device command; queue commands
AbortIO()	Abort a device command

The device-specific commands are

CMD_READ	Read one or more sectors
CMD_WRITE	Write one or more sectors
CMD_UPDATE	Write out a track buffer
CMD_CLEAR	Mark a track buffer as invalid
TD_MOTOR	Turn the motor on or off
TD_SEEK	Move the head to a specific track
TD_FORMAT	Initialize one or more tracks
TD_REMOVE	Establish a software interrupt procedure for disk removal
TD_CHANGENUM	Discover the current disk-change number
TD_CHANGESTATE	See if there is a disk present in a drive
TD_PROTSTATUS	See if a disk is write-protected

In addition to the device-specific commands listed above, the trackdisk driver has a number of extended commands. These commands are similar to their normal counterparts but have additional features: they allow you to control whether a command will be executed if the disk has been changed, and they allow you to read or write to the sector label portion of a sector.

Extended commands take a slightly larger I/O request block, which contains information that is needed only by the extended command and that is ignored by the standard form of that command. The extra information takes the form of two extra longwords at the end of the data structure. These commands are performed only if the change count is less than or equal to the one in the `iotd_Count` field of the command's I/O request block. The extended commands are listed below:

<code>ETD_READ</code>	Read one or more sectors
<code>ETD_WRITE</code>	Write one or more sectors
<code>ETD_MOTOR</code>	Turn the motor on or off
<code>ETD_UPDATE</code>	Write out a track buffer
<code>ETD_CLEAR</code>	Mark a track buffer as invalid
<code>ETD_SEEK</code>	Move the head to a specific track

Creating an I/O Request

The trackdisk device, like other devices, requires that you create an I/O request message that you pass to the device for processing. The message contains the command and several other items of control information.

Here is a program fragment that can be used to create the message block that you use for trackdisk communications. In the fragment, the routine `CreateStdIO()` is called to return a pointer to a message block. This is acceptable for the standard form of the commands. If you wish to use the extended form of the command, you will need an extended form of the request block. In place of `CreateStdIO()`, you can use the routine `CreateExtIO()`, a listing of which appears in the appendixes of the *Amiga ROM Kernel Reference Manual: Exec*.

```
struct IOStdReq *diskreq;    /* I/O request block pointer
                             * for non-extended commands */
struct IOExtTD *diskextreq; /* I/O request block pointer
                             * for extended commands */
struct Port *diskreqPort;   /* a port at which to receive replies */

diskreqPort = CreatePort("diskreq.port",0);
if(diskreqPort == 0) exit(100);    /* error in CreatePort() */
diskreq = CreateStdIO(diskreqPort);
if(diskreq == 0) { DeletePort(diskreqPort); exit(200); } /* error in CreateStdIO()
diskextreq = CreateExtIO(diskreqPort,sizeof(struct IOExtTD));
if(diskextreq == 0) { DeletePort(diskreqPort); exit(300) };
```

The routine **CreatePort()** is part of *amiga.lib*. It returns a pointer to a **Port** structure that can be used to receive replies from the trackdisk driver.

The routine **CreateStdIO()** is also in *amiga.lib*. It returns a pointer to an **IOStdReq** block that becomes the message you pass to the trackdisk driver to tell it the command to perform.

The data structure **IOExtTD** takes the form:

```
struct IOExtTD {
    struct IOStdReq iotd_Req;
    ULONG iotd_Count;
    ULONG iotd_SecLabel;
};
```

where

IOStdReq

is a standard **IORequest** block that contains fields used to transmit the standard commands (explained below).

iotd_Count

helps keep old I/O requests from being performed when the diskette has been changed. All extended commands treat as an error any case where the disk change counter is greater than **iotd_Count**. Any I/O request found with an **iotd_Count** less than the current change counter value will be returned with a characteristic error (**TDERR_DiskChange**) in the **io_Error** field of the I/O request block. This allows stale I/O requests to be returned to the user after a disk has been changed. The current disk-change counter value can be obtained by **TD_CHANGENUM**.

If the user wants extended disk I/O but does not care about disk removal, then **iotd_Count** may be set to the maximum unsigned long integer value (0xFFFFFFFF).

iotd_SecLabel

allows access to the sector identification section of the sector header.

Each sector has 16 bytes of descriptive data space available to it; the disk driver does not interpret this data. If **iotd_SecLabel** is null, then this descriptive data is ignored. If it is not null, then **iotd_SecLabel** should point to a series of 16-byte chunks (one for each sector that is to be read or written). These chunks will be written out to the sector's label region on a write or filled with the sector's label area on a read. If a **CMD_WRITE** (the standard write call) is done, then the sector label area is left unchanged.

Opening a Trackdisk Device

To gain access to a disk unit, you must first open the unit by using the system command **OpenDevice()**. A typical C-language call is shown below:

```
error = OpenDevice(TD_NAME,unit_number,disk_request_block,flags);
```

where:

TD_NAME

is a define for the null-terminated string, currently "trackdisk.device."

unit_number

is the disk unit you wish to use (defined below).

disk_request_block

is the address of an **IORequest** data structure that will later be used to communicate with the device. The **OpenDevice()** command will fill in the unit and device fields of this data structure.

flags

tell how the I/O is to be accomplished. For an **OpenDevice()** command, this field is normally set to zero.

The **unit_number** can be any value from 0 to 3. Unit 0 is the built-in 3 1/2-inch disk. Units 1 through 3 represent additional 3 1/2-inch disks that may be daisy-chained from the external disk unit connector on the back of the Amiga. The first unit (plugged directly into the Amiga) is unit 1. The second unit (plugged into unit 1), is designated as unit 2. The end-unit, farthest electrically from the Amiga, is unit 3.

The following are some common errors that may be returned from an **OpenDevice()** call.

Device in use

Some other task has already been granted exclusive use of this device.

Bad unit number

Either you have specified a unit number outside the range of 0-3 or you do not have a unit connected in the specified position.

Bad device type

You may be trying to use a 5 1/4-inch drive with the trackdisk driver. This is not supported.

Sending a Command to the Device

You send a command to this device by initializing the appropriate fields of your **IOStdReq** or **IOExtTD** and then using **SendIO()**, **DoIO()**, or **BeginIO()** to transmit the command to the device. Here is an example:

```
MotorOn()
{
    diskreq->io_Length = 1;          /* 1 says turn it on */
    diskreq->io_Command = TD_MOTOR;
    DoIO(diskreq);                  /* task sleep till command done */
    return(0);
}
```

Terminating Access to the Device

As with all exclusive-access devices, you *must* close the trackdisk device when you have finished using it. Otherwise, the system will be unable to allocate the device to any other task until the system is rebooted.

Device-specific Commands

The device-specific commands that are supported are explained below.

ETD_READ AND CMD_READ

ETD_READ obeys all of the trackdisk driver restrictions noted above. **ETD_READ** transfers data from the track buffer to the user's buffer, if and only if the disk has not been changed. If the desired sector is already in the track buffer, no disk activity is initiated. If the desired sector is not in the buffer, the track containing that sector is automatically read in. If the data in the current track buffer has been modified, it is written out to the disk before the new track is read. **CMD_READ** does not check if the disk has been changed before executing this command.

ETD_WRITE AND CMD_WRITE

ETD_WRITE obeys all of the trackdisk driver restrictions noted above. ETD_WRITE transfers data from the user's buffer to track buffer if and only if the disk has not been changed. If the track that contains this sector is already in the track buffer, no disk activity is initiated. If the desired sector is not in the buffer, the track containing that sector is automatically read in. If the data in the current track buffer has been modified, it is written out to the disk before the new track is read in for modification. CMD_WRITE does not check for disk change before performing the command.

ETD_UPDATE AND CMD_UPDATE

The Amiga trackdisk driver does not write data sectors unless it is necessary (you request that a different track be used) or until the user requests that an update be performed. This improves system speed by caching disk operations. The update commands ensure that any buffered data is flushed out to the disk. If the track buffer has not been changed since the track was read in, the update commands do nothing. In addition, ETD_UPDATE can make sure that the disk was not changed before it writes the buffer. This prevents writing the buffered data onto a different diskette.

ETD_CLEAR AND CMD_CLEAR

ETD_CLEAR marks the track buffer as invalid, forcing a reread of the disk on the next operation. ETD_UPDATE or CMD_UPDATE would be used to force data out to the disk before turning the motor off. ETD_CLEAR or CMD_CLEAR is usually used after the disk has been removed, to prevent caching of data to the new diskette. ETD_CLEAR or CMD_CLEAR will not do an update, nor will an update command do a clear. CMD_CLEAR does not check for disk change.

ETD_MOTOR AND TD_MOTOR

TD_MOTOR is called with a standard **IORequest** block. The **io_Length** field contains the requested state of the motor. A 1 will turn the motor on; a 0 will turn it off. The old state of the motor is returned in **io_Actual**. If **io_Actual** is zero, then the motor was off. Any other value implies that the motor was on. If the motor is just being turned on, the driver will delay the proper amount of time to allow the drive to come up to speed. Normally, turning the drive on is not necessary—the driver does this automatically if it receives a request when the motor is off. However, turning the motor off is the user's responsibility. In addition, the standard instructions to the user are that it is safe to remove a diskette if and only if the motor is off (that is, if the disk light is off).

TD_FORMAT

TD_FORMAT is used to write data to a track that either has not yet been formatted or has had a hard error on a standard write command. TD_FORMAT completely ignores all data currently on a track and does not check for disk change before performing the command. TD_FORMAT is called with a standard **IORequest**. The **io_Data** field must point to at least one track worth of data. The **io_Offset** field must be track aligned, and the **io_Length** field must be in units of track length (that is, $\text{NUMSECS} \times \text{TD_SECTOR}$). The driver will format the requested tracks, filling each sector with the contents of the **io_Data** field. You should do a read pass to verify the data. The command TD_FORMAT does not check whether the disk has been changed before the command is performed.

If you have a hard write error during a normal write, you may find it necessary to use the TD_FORMAT command to reformat the track as part of your error recovery process.

TD_REMOVE

TD_REMOVE is called with a standard **IORequest**. The **APTR io_Data** field points to a software interrupt structure. The driver will post this software interrupt whenever a disk is inserted or removed. To find out the current state of the disk, TD_CHANGENUM and TD_CHANGESTATE should be used. If TD_REMOVE is called with a null **io_Data** argument, then disk removal interrupts are suspended.

Status Commands

The commands that return status on the current disk in the unit are TD_CHANGENUM, TD_CHANGESTATE, and TD_PROTSTATUS.

TD_CHANGENUM

TD_CHANGENUM returns the current value of the disk-change counter (as used by the extended commands—see below). The disk change counter is incremented each time the disk is inserted or removed.

TD_CHANGESTATE

TD_CHANGESTATE returns zero if a disk is currently in the drive, and nonzero if the drive has no disk.

TD_PROTSTATUS

TD_PROTSTATUS returns nonzero if the current diskette is write-protected. All these routines return their values in **io_Actual**. These routines are safe to call from an interrupt routine (such as the software interrupt specified in TD_REMOVE). However, care should be taken when calling these routines from an interrupt. You should never **Wait()** for them to complete while in interrupt processing—it is never legal to go to sleep on the interrupt stack.

Commands for Diagnostics and Repair

Currently only one command, TD_SEEK, is provided for internal diagnostics and for disk repair.

TD_SEEK is called with a standard **IORequest**. The **io_Offset** field should be set to the (byte) offset to which the seek is to occur. TD_SEEK will not verify its position until the next read. That is, TD_SEEK only moves the heads; it does not actually read any data and it does not check to see if the disk has been changed.

Trackdisk Driver Errors

Table 7-1 is a list of error codes that can be returned by the trackdisk driver. When an error occurs, these error numbers will be returned in the **io_Error** field of your **IORequest** block.

Table 7-1: Trackdisk Driver Error Codes

Error Name	Error Number	Meaning
TDERR_NotSpecified	20	Error could not be determined
TDERR_NoSecHdr	21	Could not find sector header
TDERR_BadSecPreamble	22	Error in sector preamble
TDERR_BadSecID	23	Error in sector identifier
TDERR_BadHdrSum	24	Header field has bad checksum
TDERR_BadSecSum	25	Sector data field has bad checksum
TDERR_TooFewSecs	26	Incorrect number of sectors on track
TDERR_BadSecHdr	27	Unable to read sector header
TDERR_WriteProt	28	Disk is write-protected
TDERR_DiskChanged	29	Disk has been changed or is not currently present
TDERR_SeekError	30	While verifying seek position, found seek error
TDERR_NoMem	31	Not enough memory to do this operation
TDERR_BadUnitNum	32	Bad unit number (unit # not attached)
TDERR_BadDriveType	33	Bad drive type (not an Amiga 3 1/2 inch disk)
TDERR_DriveInUse	34	Drive already in use (only one task exclusive)
TDERR_PostReset	35	User hit reset; awaiting doom

Example Program

The following sample program exercises a few of the trackdisk driver commands.

```
#include "exec/types.h"
#include "exec/nodes.h"
#include "exec/lists.h"
#include "exec/memory.h"
#include "exec/interrupts.h"
#include "exec/ports.h"
#include "exec/libraries.h"
#include "exec/io.h"
#include "exec/tasks.h"
#include "exec/execbase.h"
#include "exec/devices.h"
```

```

#include "devices/trackdisk.h"

#define TD_READ CMD_READ
#define BLOCKSIZE TD_SECTOR

SHORT error;
struct MsgPort *diskport;
struct IOExtTD *diskreq;
BYTE diskbuffer[BLOCKSIZE];
BYTE *diskdata;
SHORT testval;

extern struct MsgPort *CreatePort();
extern struct IORequest *CreateExtIO();

ULONG diskChangeCount;

ReadCylSec(cyl, sec, hd)
SHORT cyl, sec, hd;
{
    LONG offset;

    diskreq->iotd_Req.io_Length = BLOCKSIZE;
    diskreq->iotd_Req.io_Data = (APTR)diskbuffer;
    /* show where to put the data when read */
    diskreq->iotd_Req.io_Command = ETD_READ;
    /* check that disk not changed before reading */
    diskreq->iotd_Count = diskChangeCount;

    /* convert from cylinder, head, sector to byte-offset value to get
     * right one (as dos and everyone else sees it)...*/

    /* driver reads one CYLINDER at a time (head does not move for
     * 22 sequential sector reads, or better put, head does not move for
     * 2 sequential full track reads.)
     */

    offset = TD_SECTOR * (sec + NUMSECS * hd + NUMSECS * NUMHEADS * cyl);
    diskreq->iotd_Req.io_Offset = offset;
    DoIO(diskreq);
    return(0);
}

MotorOn()
{
    /* TURN ON DISK MOTOR ... old motor state is returned in io_Actual */
    diskreq->iotd_Req.io_Length = 1;
    /* this says motor is to be turned on */
    diskreq->iotd_Req.io_Command = TD_MOTOR;
    /* do something with the motor */
    DoIO(diskreq);
    printf("\nOld motor state was: %ld", diskreq->iotd_Req.io_Actual);
}

```

```

printf("\nio_Error value was: %ld",diskreq->iotd_Req.io_Error);
return(0);
}

MotorOff()
{
printf("\n\nNow turn it off");
diskreq->iotd_Req.io_Length = 0;
/* says that motor is to be turned on */
diskreq->iotd_Req.io_Command = TD_MOTOR;
/* do something with the motor */
DoIO(diskreq);
printf("\nOld motor state was: %ld",diskreq->iotd_Req.io_Actual);
printf("\nio_Error value was: %ld",diskreq->iotd_Req.io_Error);
return(0);
}

```

```

SeekFullRange(howmany)
SHORT howmany;
{
int i;
for(i=0; i<howmany; i++)
{
diskreq->iotd_Req.io_Offset =
((NUMCYLS -1)*NUMSECS*NUMHEADS -1 ) * 512;
/* seek to cylinder 79, head 1 */
diskreq->iotd_Req.io_Command = TD_SEEK;
DoIO(diskreq);
if(diskreq->iotd_Req.io_Error != 0)
printf("\nSeek Cycle Number %ld, Error = %ld",
i, diskreq->iotd_Req.io_Error);
diskreq->iotd_Req.io_Offset = 0;
/* seek to cylinder 0, head 0 */
diskreq->iotd_Req.io_Command = TD_SEEK;
DoIO(diskreq);
if(diskreq->iotd_Req.io_Error != 0)
printf("\nSeek Cycle Number %ld, Error = %ld",
i, diskreq->iotd_Req.io_Error);
printf("\nCompleted a seek");
}
return(0);
}

```

```

main()
{
SHORT cylinder,head,sector;

diskdata = &diskbuffer[0];
/* point to first location in disk buffer */
diskport = CreatePort(0,0);
if(diskport == 0) exit(100); /* error in createport */
diskreq = (struct IOExtTD *)CreateExtIO(diskport, sizeof(struct IOExtTD));

```

```

    /* make an io request block for communicating with the disk */
if(diskreq == 0) { DeletePort(diskport); exit(200); }

    error = OpenDevice(TD_NAME,0,diskreq,0);
    /* open the device for access, unit 0 is builtin drive */
    printf("\nError value returned by OpenDevice was: %lx", error);

    /* now get the disk change value */
    diskreq->iotd_Req.io_Command = TD_CHANGENUM;
    DoIO(diskreq);
    diskChangeCount = diskreq->iotd_Req.io_Actual;
    printf("\nChange number for disk is currently %ld",diskChangeCount);

    MotorOn();
    SeekFullRange(10);
    for(cylinder=0; cylinder<80; cylinder++)    /* tracks to test */
    {
        for(head=0; head<2; head++)            /* number of heads to test */
        for(sector=0; sector<11; sector++)      /* sectors to test */
        {
            ReadCylSec(cylinder, sector, head);
            if(diskreq->iotd_Req.io_Error != 0)
                printf("\nError At Cyl=%ld, Sc=%ld, Hd=%ld, Error=%ld",
                    cylinder,sector,head,
                    diskreq->iotd_Req.io_Error);
        }
        printf("\nCompleted reading Cylinder=%ld",cylinder);
    }
    MotorOff();
    CloseDevice(diskreq);

    DeleteExtIO(diskreq, sizeof(struct IOExtTD));
    DeletePort(diskport);
} /* end of main */

```

Chapter 8

Console Device

This chapter describes how you do console (keyboard and screen) input and output on the Amiga. The console device acts like an enhanced ASCII terminal. It obeys many of the standard ANSI sequences as well as additional special sequences unique to the Amiga.

Introduction

Console I/O is tied closely to the Amiga Intuition interface; a console must be tied to a window that is already opened. From the **Window** data structure, the console device determines how many characters it can display on a line and how many lines of text it can display in a window

without clipping at any edge.

You can open the console device many times, if you wish. The result of each open call is a new console unit. AmigaDOS and Intuition see to it that only one window is currently active and its console, if any, is the only one (with a few exceptions) that receives notification of input events, such as keystrokes. Later in this chapter you will see that other Intuition events can be sensed by the console device as well.

Note: For this entire chapter the characters “<CSI>” represent the *control sequence introducer*. For output you may use either the two-character sequence “<Esc>[” or the one-byte value \$9B (hex). For input you will receive \$9B’s.

System Functions

The various system functions—such as **DoIO()**, **SendIO()**, **AbortIO()**, **CheckIO()**, and so on—operate normally. The only caveats are that **CMD_WRITE** may cause the caller to wait internally, even with **SendIO()**, and a task waiting on response from a console is at the user’s whim. If a user never reselects that window, and the console response provides the only wake-up call, that task may well sleep indefinitely.

Console I/O

The console device may be thought of as a kind of terminal. You send character streams to the console device; you also receive them from the console device. These streams may be characters or special sequences.

GENERAL CONSOLE SCREEN OUTPUT

Console character screen output (as compared to console command sequence transmission) outputs all standard printable characters (character values hex 20 thru 7E and A0 thru FF) normally. Many control characters such as **BACKSPACE** and **RETURN** are translated into their exact ANSI equivalent actions. The line-feed character is a bit different, in that it can be translated into a new-line character. The net effect is that the cursor moves to the first column of the next line whenever a <LF> is displayed. This code is set via the mode control sequences discussed under “Control Sequences for Screen Output.”

CONSOLE KEYBOARD INPUT

If you read from the console device, the keyboard inputs are preprocessed for you and you will get ASCII characters, such as "B." Most normal text-gathering programs will read from the console device in this manner. Special programs, such as word processors and music keyboard programs, will use raw input. Keys are converted via the keymap associated with the unit.

The sections below deal with the following topics:

- o Setting up for console I/O (creating an I/O request structure)
- o Writing to the console to control its behavior
- o Reading from the console
- o Closing down a console device

Creating an I/O Request

This section shows you how to set up for console I/O. Console I/O, like that used with other devices, requires that you create an I/O request message that you pass to the console device for processing. The message contains the command as well as a data area. In the data area, for a write, there will be a pointer to the stream of information you wish to write to the console. For a read, this data pointer shows where the console is to copy the data it has for you. There is also a length field that says how many characters (maximum) are to be copied either from or to the console device.

Here is a program fragment that can be used to create the message block that you use for console communications.

For *writing* to the console:

```
struct IOStdReq *consoleWriteMsg;    /* I/O request block pointer */
struct Port *consoleWritePort;      /* a port at which to receive replies*/

consoleWritePort = CreatePort("mycon.write",0);
if(consoleWritePort == 0) exit(100); /* error in createport */
consoleWriteMsg = CreateStdIO(consoleWritePort);
if(consoleWriteMsg == 0) exit(200); /* error in createstdio */
```


For *reading* from the console:

```
struct IOStdReq *consoleReadMsg;    /* I/O request block pointer */
struct Port *consoleReadPort;      /* a port at which to receive replies */

consoleReadPort = CreatePort("mycon.read",0);
if(consoleReadPort == 0) exit(300); /* error in createport */
consoleReadMsg = CreateStdIO(consoleReadPort);
if(consoleReadMsg == 0) exit(400); /* error in createstdio */
```

These fragments show two messages and ports being set up. You would use this set-up if you want to have a read command continuously queued up while using a separate message with its associated port to send control command sequences to the console. In addition, if you want to queue up multiple commands to the console, you may wish to create multiple messages (but probably just one port for receiving replied messages from the device).

Opening a Console Device

For other devices, you normally use **OpenDevice()** to pass an uninitialized **IORequest** block to the device. For a console device, a slightly different method is used. You must have initialized two fields in the request block; namely, the data pointer and the length field. Here is a subroutine that can be used to open a console device (attach it to an existing window). It assumes that `intuition.library` is already open, a window has also been opened, and this new console is to be attached to the open window.

```

/* this function returns a value of 0 if the console
 * device opened correctly and a nonzero value
 * (the error returned from OpenDevice) if there was an error.
 */

OpenConsole(writerequest,readrequest>window)
    struct IOStdReq *writerequest;
    struct IOStdReq *readrequest;
    struct Window *window;
    {
        int error;
        writerequest->io_Data = (APTR) window;
        writerequest->io_Length = sizeof(*window);
        error = OpenDevice("console.device", 0, writerequest, 0);
        readrequest->io_Device = writerequest->io_Device;
        readrequest->io_Unit = writerequest->io_Unit;
        /* clone required parts of the request */
        return(error);
    }

```

Notice that this routine opens the console using one I/O request (write), then copies the write request values into the read request. This assures that both input and output go to the same console device.

SENDING A CHARACTER STREAM TO THE CONSOLE DEVICE

To perform console I/O, you fill in fields of the console I/O standard request and pass this block to the console device using one of the normal I/O functions. When the console device has completed the action, the device returns the message block to the port you have designated within the message itself. The function **CreateStdIO()** initializes the message to contain the address of the **ReplyPort**.

The following subroutines use the **IOStdReq** created above. Note that the **IOStdReq** itself contains a pointer to the unit with which it is communicating. Thus, a single function can be used to communicate with multiple consoles.

```
/* output a single character to a specified console */
```

```
ConPutChar(request,character)
```

```
struct IOStdReq *request;
```

```
char character;
```

```
{  
    request->io_Command = CMD_WRITE;  
    request->io_Data = &character;  
    request->io_Length = 1;  
    DoIO(request);  
    return;  
}
```

```
/* output a stream of known length to a console */
```

```
ConWrite(request,string,length)
```

```
struct IOStdReq *request;
```

```
char *string;
```

```
int length;
```

```
{  
    request->io_Command = CMD_WRITE;  
    request->io_Data = string;  
    request->io_Length = length;  
    DoIO(request);  
    return;  
}
```

```
/* output a NULL-terminated string of characters to a console */
```

```
ConPutStr(request,string)
```

```
struct IOStdReq *request;
```

```
char *string;
```

```
{  
    request->io_Command = CMD_WRITE;  
    request->io_Data = string;  
    request->io_Length = -1;    /* tells console to end when it sees a  
                               * terminating zero on the string. */  
    DoIO(request);
```

```
    return;  
}
```

Control Sequences for Screen Output

Table 8-1 lists the functions that the console device supports, along with the character stream that you must send to the console to produce the effect. Where the function table indicates multiple characters, it is more efficient to use the **ConWrite()** function rather than **ConPutChar()** because it avoids the overhead of transferring the message block multiple times. The table below uses the second form of <CSI>, that is, the hex value 9B, to minimize the number of characters to be transmitted to produce a function.

In table 8-1, if an item is enclosed in square brackets, it is optional and may be omitted. For example, for INSERT [N] CHARACTERS the value for N or M is shown as optional. The console device responds to such optional items by treating the value of N as if it is not specified. The value of N or M is always a decimal number, having one or more ASCII digits to express its value.

Table 8-1: Console Control Sequences

Command	Sequence of Characters (in Hexadecimal Form)
BACKSPACE (move left one column)	08
LINE FEED (move down one text line as specified by the mode function below)	0A
VERTICAL TAB (move up one text line)	0B
FORM FEED (clear the console's screen)	0C
CARRIAGE RETURN (move to first column)	0D
SHIFT IN (undo SHIFT OUT)	0E
SHIFT OUT (set MSB of each character before displaying)	0F
ESC (escape; can be part of the control sequence introducer)	1B
CSI (control sequence introducer)	
RESET TO INITIAL STATE	1B 63
INSERT [N] CHARACTERS (Inserts one or more spaces, shifting the remainder of the line to the right.)	9B [N] 40
CURSOR UP [N] CHARACTER POSITIONS (default = 1)	9B [N] 41
CURSOR DOWN [N] CHARACTER POSITIONS (default = 1)	9B [N] 42

CURSOR FORWARD [N] CHARACTER POSITIONS (default = 1)	9B [N] 43
CURSOR BACKWARD [N] CHARACTER POSITIONS (default = 1)	9B [N] 44
CURSOR NEXT LINE [N] (to column 1)	9B [N] 45
CURSOR PRECEDING LINE [N] (to column 1)	9B [N] 46
MOVE CURSOR TO ROW; COLUMN where N is row, M is column, and semicolon (hex 3B) must be present as a separator, or if row is left out, so the console device can tell that the number after the semicolon actually represents the column number.	9B [N] [3B N] 48
ERASE TO END OF DISPLAY	9B 4A
ERASE TO END OF LINE	9B 4B
INSERT LINE (above the line containing the cursor)	9B 4C
DELETE LINE (remove current line, move all lines up one position to fill gap, blank bottom line)	9B 4D
DELETE CHARACTER [N] (that cursor is sitting on and to the right if [N] is specified)	9B [N] 50
SCROLL UP [N] LINES (Remove line(s) from top of screen, move all other lines up, blanks [N] bottom lines)	9B [N] 53
SCROLL DOWN [N] LINES (Remove line(s) from bottom of screen, move all other lines down, blanks [N] top lines)	9B [N] 54
SET MODE (cause LINEFEED to respond as RETURN-LINEFEED)	9B 32 30 68
RESET MODE (cause LINEFEED to respond only as LINEFEED)	9B 32 30 6C
DEVICE STATUS REPORT (cause console to insert into your read-stream a CURSOR POSITION REPORT; see "Reading from the Console" for more information)	9B 36 6E
SELECT GRAPHIC RENDITION <style>;<fg>;<bg>6D (select text style foreground color, background color) (See the note below.)	See note below.

Note: For SELECT GRAPHIC RENDITION, any number of parameters, in any order, are valid. They are separated by semicolons. The parameters follow:

<style> =

- 0 Plain text
- 1 Bold-face
- 3 Italic
- 4 Underscore
- 7 Inverse-video

<fg> =

- 30 - 37 Selecting system colors 0-7 for foreground.
Transmitted as two ASCII characters.

<bg> =

- 40 - 47 selecting system colors 0-7 for background.
Transmitted as two ASCII characters.

For example, to select bold face, with color 3 as foreground and color 0 as background, send the sequence:

```
9B 31 3B 33 33 3B 34 30 6D
```

representing the ASCII sequence:

```
"<CSI>1;33;40m"
```

where <CSI> is the control sequence introducer, here used as the single-character value 9B hex.

The sequences in table 8-2 are not ANSI standard sequences; they are private Amiga sequences.

In these command descriptions, length, width, and offset are comprised of one or more ASCII digits, defining a decimal value.

Table 8-2: Amiga Console-control Sequences

| Command | Sequence of Characters
(in Hexadecimal Form) |
|---|---|
| SET PAGE LENGTH (in character raster lines, causes console to recalculate, using current font, how many text lines will fit on the page). | 9B <length> 74 |
| SET LINE LENGTH (in character positions, using current font, how many characters should be placed on each line). | 9B <width> 75 |
| SET LEFT OFFSET (in raster columns, how far from the left of the window should the text begin). | 9B <offset> 78 |
| SET TOP OFFSET (in raster lines, how far from the top of the window's RastPort should the topmost line of the character begin). | 9B <offset> 79 |
| SET RAW EVENTS—see the separate topic “Selecting Raw Input Events” below for more details. | |
| RESET RAW EVENTS—see “Selecting Raw Input Events” below. | |
| SET CURSOR RENDITION - make the cursor visible or invisible: | |
| Invisible: | 9B 30 20 70 |
| Visible: | 9B 20 70 |
| WINDOW STATUS REQUEST - ask the console device to tell you the current bounds of the window, in upper and lower row and column character positions. (User may have resized or repositioned it.) See “Window Bounds Report” below. | 9B 30 20 71 |

Note: The console device normally handles the SET PAGE LENGTH, SET LINE LENGTH, SET LEFT OFFSET, and SET TOP OFFSET functions automatically. To allow it to do so again after setting your own values, you can send the function without a parameter.

Examples

Move cursor right by 1:

Character string equivalents: <CSI>C or <CSI>1C
Numeric (hex) equivalents: 9B 43 9B 31 43

Move cursor right by 20:

Character string equivalent: <CSI>20C
Numeric (hex) equivalent: 9B 32 30 43

Move cursor to upper left corner (home):

Character string equivalents:
<CSI>H or
<CSI>1;1H or
<CSI>;1H or
<CSI>1;H

Numeric (hex) equivalents:
9B 48
9B 31 3B 31 48
9B 3B 31 48
9B 31 3B 48

Move cursor to the fourth column of the first line of the window:

Character string equivalents:
<CSI>1;4H or
<CSI>;4H

Numeric (hex) equivalents:
9B 31 3B 34 48
9B 3B 34 48

Clear the screen:

Character string equivalents:

<FF> or CTRL-L {clear screen character} or
<CSI>H<CSI>J {home and clear to end of screen} or

Numeric (hex) equivalents:

0C
9B 48 9B 4A

READING FROM THE CONSOLE

Reading input from the console device returns an ANSI 3.64 standard byte stream. This stream may contain normal characters and/or RAW input event information. You may also request other RAW input events using the SET RAW EVENTS and RESET RAW EVENTS control sequences discussed below. See "Selection of Raw Input Events."

The following subroutines are useful for setting up for console reads. Only a single-character-at-a-time version is shown here.

Note: This example does not illustrate the fact that a request for more than one character can be satisfied by only one, thus requiring you to look at **io_Actual**.

```
/* queue up a read request to a console, show where to put the character when ready
 * to be returned. Most efficient if this is called right after console is opened */
```

```
QueueRead(request,wheretoto)
struct IOStdReq *request;
char *wheretoto;
{
    request->io_Command = CMD_READ;
    request->io_Data = wheretoto;
    request->io_Length = 1;
    SendIO(request);
    return;
}
```

```
/* see if there is a character to read. If none, don't wait,
 * come back with a value of -1 */
```

```
int
ConMayGetChar(consolePort,request,wheretoto)
struct Port *consolePort;
```

```

struct IOStdReq *request;
char *whereto;
{
    register temp;

    if ( GetMsg(consolePort) == NULL ) return(-1);
    temp = *whereto;
    QueueRead(request,whereto);
    return(temp);
}

/* go and get a character; put the task to sleep if
 * there isn't one present */

UBYTE
ConGetChar(consolePort,request,whereto)
struct IOStdReq *request;
struct Port *consolePort;
char *whereto;
{
    register temp;
    while((GetMsg(consolePort) == NULL)) WaitPort(consolePort);
    temp = *whereto;    /* get the character */
    QueueRead(request,whereto);
    return(temp);
}

```

INFORMATION ABOUT THE READ-STREAM

For the most part, keys whose keycaps are labeled with ANSI standard characters will ordinarily be translated into their ASCII-equivalent character by the console device through the use of its keymap. A separate section in this chapter has been dedicated to the method used to establish a keymap and the internal organization of the keymap.

For keys other than those with normal ASCII equivalents, an escape sequence is generated and inserted into your input stream. For example, in the default state (no raw input events selected) the function and arrow keys will cause the sequences shown in table 8-3 to be inserted in the input stream.

Table 8-3: Special Key Report Sequences

| Key | Unshifted Sends | Shifted Sends | |
|-------------|-----------------|---------------|-------------------|
| F1 | <CSI>0~ | <CSI>10~ | |
| F2 | <CSI>1~ | <CSI>11~ | |
| F3 | <CSI>2~ | <CSI>12~ | |
| F4 | <CSI>3~ | <CSI>13~ | |
| F5 | <CSI>4~ | <CSI>14~ | |
| F6 | <CSI>5~ | <CSI>15~ | |
| F7 | <CSI>6~ | <CSI>16~ | |
| F8 | <CSI>7~ | <CSI>17~ | |
| F9 | <CSI>8~ | <CSI>18~ | |
| F10 | <CSI>9~ | <CSI>19~ | |
| HELP | <CSI>?~ | <CSI>?~ | (same) |
| Arrow keys: | | | |
| Up | <CSI>A | <CSI>T | |
| Down | <CSI>B | <CSI>S | |
| Left | <CSI>D | <CSI> A | (notice the space |
| Right | <CSI>C | <CSI> @ | after <CSI>) |

CURSOR POSITION REPORT

If you have sent the DEVICE STATUS REPORT command sequence, the console device returns a cursor position report into your input stream. It takes the form:

```
<CSI><row>;<column>R
```

For example, if the cursor is at column 40 and row 12, here are the ASCII values you receive in a stream:

```
9B 34 30 3B 31 32 52
```

WINDOW BOUNDS REPORT

A user may have either moved or resized the window to which your console is bound. By issuing a WINDOW STATUS REPORT to the console, you can read the current position and size in the input stream. This window bounds report takes the following form:

```
<CSI>1;1;<bottom margin>;<right margin>r
```

Note that the top and left margins are always 11 for the Amiga. The bottom and right margins give you the window row and column dimensions as well. For a window that holds 20 lines with 60 characters per line, you will receive the following in the input stream:

```
9B 31 3B 31 3B 32 30 3B 36 30 20 72
```

SELECTING RAW INPUT EVENTS

If the keyboard information—including “cooked” keystrokes—does not give you enough information about input events, you can request additional information from the console driver.

The command to SET RAW EVENTS is formatted as:

```
” <CSI>[event-types-separated-by-semicolons]{”
```

If, for example, you need to know when each key is pressed and released you would request “RAW keyboard input.” This is done by writing “<CSI>1{” to the console. In a single SET RAW EVENTS request, you can ask the console to set up for multiple event types at one time. You must send multiple numeric parameters, separating them by semicolons (;). For example, to ask for gadget pressed, gadget released, and close gadget events, write “<CSI>7;8;11{” (all as ASCII characters, without the quotes).

You can reset, that is, delete from reporting, one or more of the raw input event types by using the RESET RAW EVENTS command, in the same manner as the SET RAW EVENTS was used to establish them in the first place. This command stream is formatted as:

```
<CSI>[event-types-separated-by-semicolons]}
```

So, for example, you could reset all of the events set in the above example by transmitting the command sequence: “<CSI>7;8;11}.” Table 8-4 is a list of the valid raw input event types.

Table 8-4: Raw Input Event Types

| Request Number | Description | |
|----------------|---------------------|---|
| 0 | No-op | Used internally |
| 1 | RAW keyboard input | Intuition swallows all except the select button |
| 2 | RAW mouse input | |
| 3 | Event | Sent whenever your window is made active |
| 4 | Pointer position | |
| 5 | (unused) | |
| 6 | Timer | |
| 7 | Gadget pressed | |
| 8 | Gadget released | |
| 9 | Requester activity | |
| 10 | Menu numbers | |
| 11 | Close Gadget | |
| 12 | Window resized | |
| 13 | Window refreshed | |
| 14 | Preferences changed | |
| 15 | Disk removed | |
| 16 | Disk inserted | |

Complex Input Event Reports

If you select any of these events you will start to get information about the events in the following form:

```
<CSI><class>;<subclass>;<keycode>;<qualifiers>;<x>;<y>;
<seconds>;<microseconds>|
```

where

<CSI>

is a one-byte field. It is the “control sequence introducer”, 9B in hex.

<class>

is the RAW input event type, from the above table.

<subclass>

is usually 0. If the mouse is moved to the right controller, this would be 1.

<keycode>

indicates which key number was pressed (see figure 8-1 and table 8-6). This field can also be used for mouse information.

<qualifiers>

indicates the state of the keyboard and system. The qualifiers are defined as shown in table 8-5.

Table 8-5: Input Event Qualifiers

| Bit | Mask | Key | |
|-----|------|-------------------------|---|
| 0 | 0001 | Left shift | |
| 1 | 0002 | Right shift | |
| 2 | 0004 | Caps Lock | Associated keycode is special; see below. |
| 3 | 0008 | Ctrl | |
| 4 | 0010 | Left Alt | |
| 5 | 0020 | Right Alt | |
| 6 | 0040 | Left Amiga key pressed | |
| 7 | 0080 | Right Amiga key pressed | |
| 8 | 0100 | Numeric pad | |
| 9 | 0200 | Repeat | |
| 10 | 0400 | Interrupt | Not currently used. |
| 11 | 0800 | Multi-broadcast | This window (active one) or all windows. |
| 12 | 1000 | Left mouse button | |
| 13 | 2000 | Right mouse button | |
| 14 | 4000 | Middle mouse button | (Not available on standard mouse) |
| 15 | 8000 | Relative mouse | Indicates mouse coordinates are relative, not absolute. |

The Caps Lock key is handled in a special manner. It generates a keycode only when it is pressed, not when it is released. However, the up/down bit (80 hex) is still used and reported. If pressing the Caps Lock key causes the LED to light, keycode 62 (Caps Lock pressed) is sent. If pressing the Caps Lock key extinguishes the LED, keycode 190 (Caps Lock released) is sent. In effect, the keyboard reports this key as held down until it is struck again.

The <x> and <y> fields are filled by some classes with an Intuition address: $x \ll 16 + y$.

The <seconds> and <microseconds> fields contain the system time stamp taken at the time the event occurred. These values are stored as long-words by the system.

With RAW keyboard input selected, keys will no longer return a simple one-character “A” to “Z” but will instead return raw keycode reports of the form:

```
<CSI>1;0;<keycode>;<qualifiers>;0;0;<seconds>;<microseconds>|
```

For example, if the user pressed and released the “B” key with the left Shift and right Amiga keys also pressed, you might receive the following data:

```
<CSI>1;0;35;129;0;0;23987;99|
<CSI>1;0;163;129;0;0;24003;18|
```

The <keycode> field is an ASCII decimal value representing the key pressed or released. Adding 128 to the pressed key code will result in the released keycode. Figure 4-1 lets you convert quickly from a key to its keycode. The tables let you convert quickly from a keycode to a key.

| | | | | | | | | | | | | | | | | | |
|-------------|-----------------|----------|----------|----------|----------|----------|----------|-----------|----------|-----------|-----------|-------------|------------------|------------|-------------|---------|---------|
| ESC
45 | F1
50 | F2
51 | F3
52 | F4
53 | F5
54 | F6
55 | F7
56 | F8
57 | F9
58 | F10
59 | DEL
46 | | | | | | |
| ~
00 | 1
01 | @
02 | 3
03 | \$
04 | %
05 | ^
06 | &
07 | *
08 | 9
09 | 0
0A | -
0B | =
0C | BACK SPACE
41 | 7
3D | 8
3E | 9
3F | |
| TAB
42 | Q
10 | W
11 | E
12 | R
13 | T
14 | Y
15 | U
16 | I
17 | O
18 | P
19 | [
1A |]
1B | 44 | HELP
5F | 4
2D | 5
2E | 6
2F |
| CTRL
63 | CAPS LOCK
62 | A
20 | S
21 | D
22 | F
23 | G
24 | H
25 | J
26 | K
27 | L
28 | ;
29 | "
2A | RETURN
2B | 4C | 1
1D | 2
1E | 3
1F |
| SHIFT
60 | 30 | Z
31 | X
32 | C
33 | V
34 | B
35 | N
36 | M
37 | <
38 | >
39 | ?
3A | SHIFT
61 | 4F | 4E | 0
0F | 3C | |
| ALT
64 | A
66 | 40 | | | | | | ALT
67 | 65 | | | | 4D | -
4A | ENTER
43 | | |

Figure 8-1: The Amiga Keyboard, Showing Keycodes in Hex

The default values given correspond to the values the console device will return when these keys are pressed and the keycaps as shipped with the standard American keyboard.

Table 8-6: System Default Console Key Mapping

| Raw Key Number | Keycap Legend | Unshifted Default Value | Shifted Default Value |
|----------------|---------------|-------------------------|-----------------------|
| 00 | ‘ ~ | ‘ (Accent grave) | ~ (tilde) |
| 01 | 1 ! | 1 | ! |
| 02 | 2 @ | 2 | @ |
| 03 | 3 # | 3 | # |
| 04 | 4 \$ | 4 | \$ |
| 05 | 5 % | 5 | % |
| 06 | 6 ^ | 6 | ^ |
| 07 | 7 & | 7 | & |
| 08 | 8 * | 8 | * |
| 09 | 9 (| 9 | (|
| 0A | 0) | 0 |) |
| 0B | - _ | - (Hyphen) | _ (Underscore) |
| 0C | = + | = | + |
| 0D | \ | \ | |
| 0E | | (undefined) | |
| 0F | 0 | 0 | 0 (Numeric pad) |
| 10 | Q | q | Q |
| 11 | W | w | W |
| 12 | E | e | E |
| 13 | R | r | R |
| 14 | T | t | T |
| 15 | Y | y | Y |
| 16 | U | u | U |
| 17 | I | i | I |
| 18 | O | o | O |
| 19 | P | p | P |
| 1A | [{ | [| { |
| 1B |] } |] | } |
| 1C | | (undefined) | |
| 1D | 1 | 1 | 1 (Numeric pad) |
| 1E | 2 | 2 | 2 (Numeric pad) |
| 1F | 3 | 3 | 3 (Numeric pad) |
| 20 | A | a | A |
| 21 | S | s | S |
| 22 | D | d | D |
| 23 | F | f | F |
| 24 | G | g | G |

| Raw Key Number | Keycap Legend | Unshifted Default Value | Shifted Default Value |
|----------------|---------------|-------------------------|-----------------------|
| 25 | H | h | H |
| 26 | J | j | J |
| 27 | K | k | K |
| 28 | L | l | L |
| 29 | ; : | ; | : |
| 2A | ' " | ' (single quote) | " |
| 2B | | (RESERVED) | (RESERVED) |
| 2C | | (undefined) | |
| 2D | 4 | 4 | 4 (Numeric pad) |
| 2E | 5 | 5 | 5 (Numeric pad) |
| 2F | 6 | 6 | 6 (Numeric pad) |
| 30 | | (RESERVED) | (RESERVED) |
| 31 | Z | z | Z |
| 32 | X | x | X |
| 33 | C | c | C |
| 34 | V | v | V |
| 35 | B | b | B |
| 36 | N | n | N |
| 37 | M | m | M |
| 38 | , < | , (comma) | < |
| 39 | . > | . (period) | > |
| 3A | / ? | / | ? |
| 3B | | (undefined) | |
| 3C | . | . | . (Numeric pad) |
| 3D | 7 | 7 | 7 (Numeric pad) |
| 3E | 8 | 8 | 8 (Numeric pad) |
| 3F | 9 | 9 | 9 (Numeric pad) |
| 40 | (Space bar) | 20 | 20 |
| 41 | Back Space | 08 | 08 |
| 42 | Tab | 09 | 09 |
| 43 | Enter | 0D | 0D (Numeric pad) |
| 44 | Return | 0D | 0D |
| 45 | Esc | 1B | 1B |
| 46 | Del | 7F | 7F |
| 47 | | (undefined) | |
| 48 | | (undefined) | |
| 49 | | (undefined) | |
| 4A | | - | - (Numeric Pad) |
| 4B | | (undefined) | |

| Raw Key Number | Keycap Legend | Unshifted Default Value | Shifted Default Value |
|----------------|--|-------------------------|-----------------------|
| 4C | Up arrow | <CSI>A | <CSI>T |
| 4D | Down arrow | <CSI>B | <CSI>S |
| 4E | Forward arrow
(note blank space after <CSI>) | <CSI>C | <CSI> A |
| 4F | Backward arrow
(note blank space after <CSI>) | <CSI>D | <CSI> @ |
| 50 | F1 | <CSI>0~ | <CSI>10~ |
| 51 | F2 | <CSI>1~ | <CSI>11~ |
| 52 | F3 | <CSI>2~ | <CSI>12~ |
| 53 | F4 | <CSI>3~ | <CSI>13~ |
| 54 | F5 | <CSI>4~ | <CSI>14~ |
| 55 | F6 | <CSI>5~ | <CSI>15~ |
| 56 | F7 | <CSI>6~ | <CSI>16~ |
| 57 | F8 | <CSI>7~ | <CSI>17~ |
| 58 | F9 | <CSI>8~ | <CSI>18~ |
| 59 | F10 | <CSI>9~ | <CSI>19~ |
| 5A | | (undefined) | |
| 5B | | (undefined) | |
| 5C | | (undefined) | |
| 5D | | (undefined) | |
| 5E | | (undefined) | |
| 5F | HELP | <CSI>?~ | <CSI>?~ |

| Raw
Key
Number | Function or
Keycap
Legend | |
|-------------------------------|--|---|
| 60 | Shift (left of space bar) | |
| 61 | Shift (right of space bar) | |
| 62 | Caps Lock | |
| 63 | Ctrl | |
| 64 | (Left) Alt | |
| 65 | (Right) Alt | |
| 66 | Amiga (left of space bar) | Close Amiga |
| 67 | Amiga (right of space bar) | Open Amiga |
| 68 | Left mouse button
(not converted) | Inputs are only for the
mouse connected to Intuition,
currently "gameport" one. |
| 69 | Right mouse button
(not converted) | |
| 6A | Middle mouse button
(not converted) | |
| 6B | (undefined) | |
| 6C | (undefined) | |
| 6D | (undefined) | |
| 6E | (undefined) | |
| 6F | (undefined) | |

| Raw Key Number | Function |
|----------------|--|
| 70-7F | (undefined) |
| 80-F8 | Up transition (release or unpress key of one of the above keys) (80 for 00, F8 for 7F) |
| F9 | Last keycode was bad
(was sent in order to resynchronize) |
| FA | Keyboard buffer overflow |
| FB | (undefined, reserved for
keyboard processor catastrophe) |
| FC | Keyboard selftest failed |
| FD | Power-up key stream start.
Keys pressed or stuck at power-up
will be sent between FD and FE. |
| FE | Power-up key stream end |
| FF | (undefined, reserved) |
| FF | Mouse event, movement only,
no button change (not converted) |

Notes about the preceding table:

- 1) “(undefined)” indicates that the current keyboard design should not generate this number. If you are using **SetKeyMap()** to change the key map, the entries for these numbers must still be included.
- 2) “(not converted)” refers to mouse button events. You must use the sequence “<CSI>2{” to inform the console driver that you wish to receive mouse events; otherwise these will not be transmitted.
- 3) “(RESERVED)” indicates that these keycodes have been reserved for non-US keyboards. The “2B” code key will be between the double-quote(”) and Return keys. The “30” code key will be between the Shift and “Z” keys.

Keymapping

The Amiga has the capability of mapping the keyboard in any manner that you wish. In other computers, this capability is normally provided through the use of “keyboard enhancers.” In the Amiga, however, the capability is already present and the vectors that control the remapping are user-accessible.

The functions called **AskKeyMap()** and **SetKeyMap()** each deal with a set of eight longword pointers, known as the **KeyMap** data structure. The **KeyMap** data structure is shown below.

```
struct KeyMap {
    UBYTE *km_LoKeyMapTypes;
    ULONG *km_LoKeyMap;
    UBYTE *km_LoCapsable;
    UBYTE *km_LoRepeatable;
    UBYTE *km_HiKeyMapTypes;
    ULONG *km_HiKeyMap;
    UBYTE *km_HiCapsable;
    UBYTE *km_HiRepeatable;
};
```

The function **AskKeyMap()** shown below does not return a pointer to a table of pointers to currently assigned key mapping. Instead, it *copies* the current set of pointers to a user-designated area of memory. **AskKeyMap()** returns a TRUE/FALSE value that says whether or not the function succeeded.

The function **SetKeyMap()**, also shown below, copies the designated key map data structure to the console device. Thus this routine is complementary to **AskKeymap()** in that it can restore an original key mapping as well as establish a new one.

```
/* this include file is needed as well as
 * other normal console includes */

#include "devices/keymap.h"

int AskKeyMap(request,keymap)
    struct IOStdReq *request;
    struct KeyMap *keymap;
{
    int i;
    request->io_Command = CD_ASKKEYMAP;
    request->io_Length = sizeof(struct KeyMap);
    request->io_Data = keymap; /* where to put it */
    DoIO(request);
    i = request->io_Error;
    if(i) return(FALSE);
    else return(TRUE); /* if no error, it worked. */
}
```

```

int SetKeyMap(request,keymap)
    struct IOStdReq *request;
    struct KeyMap *keymap;
{
    int i;
    request->io_Command = CD_SETKEYMAP;
    request->io_Length = sizeof(struct KeyMap);
    request->io_Data = keymap;    /* where to get it */
    DoIO(request);
    i = request->io_Error;
    if(i) return(FALSE);
    else return(TRUE);    /* if no error, it worked. */
}

```

As a prelude to the following material, note that the Amiga keyboard transmits raw key information to the computer in the form of a key position and a transition. Figure 8-1 shows a physical layout of the keys and the hexadecimal number that is transmitted to the system when a key is pressed. When the key is released, its value, plus hexadecimal 80, is transmitted to the computer. The key mapping described herein refers to the translation from this raw key transmission into console device output to the user.

The low key map provides translation of the key values from hex 00-3F; the high key map provides translation of key values from hex 40-67. Raw output from the keyboard for the low key map does not include the space bar, Tab, Alt, Ctrl, arrow keys, and several other keys (see figure 8-2 and table 8-7).

| | | | | | | | | | | | | | | | | |
|----|----|----|----|-----|----|----|----|----|----|---|----|----|----|----|----|----|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 3D | 3E | 3F |
| | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | | 2D | 2E | 2F |
| | | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 2A | | 1D | 1E | 1F |
| | | | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 3A | | 0F | | 3C |
| ~ | 1! | 2@ | 3# | 4\$ | 5% | 6^ | 7& | 8* | 9(| 0) | -_ | =+ | \ | 7 | 8 | 9 |
| | qQ | wW | eE | rR | tT | yY | uU | iI | oO | pP | [{ |]} | | 4 | 5 | 6 |
| | aA | sS | dD | fF | gG | hH | jJ | kK | lL | ; | : | " | | 1 | 2 | 3 |
| | | zZ | xX | cC | vV | bB | nN | mM | ,< | . <td>/?</td> <td></td> <td></td> <td>0</td> <td></td> <td>.</td> | /? | | | 0 | | . |

Figure 8-2: Low Key Map Translation Table

Table 8-7: High Key Map Hex Values

| Key Number | Function or Keycap Legend |
|------------|---------------------------|
| 40 | Space |
| 41 | Backspace |
| 42 | Tab |
| 43 | Enter |
| 44 | Return |
| 45 | Escape |
| 46 | Delete |
| 4A | Numeric Pad - character |
| 4C | Cursor Up |
| 4D | Cursor Down |
| 4E | Cursor Forward |
| 4F | Cursor Backward |
| 50-59 | Function keys F1-F10 |
| 5F | Help |
| 60 | Left Shift |
| 61 | Right Shift |
| 62 | Caps Lock |
| 63 | Control |
| 64 | Left Alt |
| 65 | Right Alt |
| 66 | Left Amiga |
| 67 | Right Amiga |

The keymap table for the low and high keymaps consists of 4-byte entries, one per hex keycode. These entries are interpreted in one of two possible ways:

- o As four separate bytes, specifying how the key is to be interpreted when pressed alone, with one qualifier, with another qualifier, or with both qualifiers (where a qualifier is one of three possible keys: Ctrl, Alt, or Shift).
- o As a longword containing the address of a string descriptor, where a string of hex digits is to be output when this key is pressed. If a string is to be output, any combination of qualifiers may affect the string that may be transmitted.

Note: The keymap table *must* begin aligned on a word boundary. Each entry is four bytes long, thereby maintaining word alignment throughout the table. This is necessary because some of the entries may be longword addresses and *must* be aligned properly for the 68000.

ABOUT QUALIFIERS

As you may have noticed, there are three possible qualifiers, but only a 4-byte space in the table for each key. This does not allow space to describe what the computer should output for all possible combinations of qualifiers. This problem is solved by only allowing all three qualifiers to affect the output at the same time in string mode. Here is how that works.

For “vanilla” keys, such as the alphabetic keys, use the 4 bytes to represent the data output for the key alone, Shifted key, Alt’ed key, and Shifted-and-Alt’ed key. Then for the Ctrl-key-plus-vanilla-key, use the code for the key alone with bits 6 and 5 set to 0.

For other keys, such as the Return key or Esc key, the qualifiers specified in the keytypes table (up to two) are the qualifiers used to establish the response to the key. This is done as follows. In the keytypes table, the values listed for the key types are those listed for the qualifiers in *devices/keymap.h* and *devices/keymap.i*. Specifically, these qualifier equates are:

| | |
|-------------|------|
| KC_NOQUAL | 0x00 |
| KCF_SHIFT | 0x01 |
| KCF_ALT | 0x02 |
| KCF_CONTROL | 0x04 |
| KC_VANILLA | 0x07 |
| KCF_DOWNUP | 0x08 |
| KCF_STRING | 0x40 |

As shown above, the qualifiers for the various types of keys occupy specific bit positions in the key types control byte.

In assembly code, a keymap table entry looks like this:

```
SOME_KEY:  
    DC.B  VALUE_1, VALUE_2, VALUE_3, VALUE_4
```

Table 8-8 shows how to interpret the keymap for various combinations of the qualifier bits.

0x00 00000000
 0x10 00000000
 0x40 00000000
 0x80 00000000

Table 8-8: Keymap Qualifier Bits

| If Keytype is: | Then value in this position in the keytable is output when the key is pressed along with: | | | |
|----------------------------|---|------|-------|--------|
| 0x00 KC_NOQUAL | - | - | - | alone |
| 0x01 KCF_SHIFT | - | - | Shift | alone |
| 0x02 KCF_ALT | - | - | Alt | alone |
| 0x04 KCF_CONTROL | - | - | Ctrl | alone |
| 0x03 KCF_ALT+KCF_SHIFT | Shift+Alt | Alt | Shift | alone |
| 0x06 KCF_CONTROL+KCF_ALT | Ctrl+Alt | Ctrl | Alt | alone |
| 0x05 KCF_CONTROL+KCF_SHIFT | Ctrl+Shift | Ctrl | Shift | alone |
| 0x07 KC_VANILLA | Shift+Alt | Alt | Shift | alone* |

* Special case—Ctrl key, when pressed with one of the alphabet keys and certain others, is to output key-alone value with the bits 6 and 5 set to zero.

KEYTYPE TABLE ENTRIES

The vectors named `km_LoKeyTypes` and `km_HiKeyTypes` contain one byte per raw key code. This byte defines the entry type that is made in the key table by a set of bit positions.

Possible key types are:

- o Any of the qualifier groupings noted above
- o `KCF_STRING` + any combination of `KCF_SHIFT`, `KCF_ALT`, `KCF_CONTROL` (or `KC_NOQUAL`) if the result of pressing the key is to be a stream of bytes (and key-with-one-or-more-qualifiers is to be one or more alternate streams of bytes).

Any key can be made to output up to eight unique byte streams if `KCF_STRING` is set in its keytype. The only limitation is that the total length of all of the strings assigned to a key be within the “jump range” of a single byte increment. See the “String-Output Keys” section below for more information.

The low keytype table covers the raw keycodes from hex 00-3F and contains one byte per key-code. Therefore this table contains 64 (decimal) bytes. The high keytype table covers the raw keycodes from hex 40-67 and contains 38 (decimal) bytes.

STRING-OUTPUT KEYS

When a key is to output a string, the keymap table contains the address of a string descriptor in place of a 4-byte mapping of a key as shown above. Here is a partial table for a new high key map table that contains only three entries thus far. The first two are for the space bar and the backspace key; the third is for the tab key, which is to output a string that says “[TAB].” An alternate string, “[SHIFTED-TAB],” is also to be output when a shifted TAB key is pressed.

newHiMapTypes:

| | | |
|------|-----------------------|---------|
| DC.B | KCF_ALT,KC_NOQUAL, | |
| DC.B | KCF_STRING+KCF_SHIFT, | |
| | ... | ;(more) |

newHiMap:

| | | |
|------|---------------|-------------------------------|
| DC.B | 0,0,\$A0,\$20 | ;space bar, and Alt-space bar |
| DC.B | 0,0,0,\$08 | ;Back Space key only |
| DC.L | newkey42 | ;new definition for string to |
| | ... | ;output for Tab key |
| | | ;(more) |

newkey42:

| | | |
|------|--------------------|--------------------------------|
| DC.B | new42ue - new42us | ;length of the |
| | | ;unshifted string |
| DC.B | new42us - newkey42 | |
| | | ;number of bytes from start of |
| | | ;string descriptor to start of |
| | | ;this string |
| DC.B | new42se - new42ss | |
| | | ;length of the shifted string |
| DC.B | new42ss - newkey42 | |
| | | ;number of bytes from start of |
| | | ;string descriptor to start of |
| | | ;this string |

new42us:

| | |
|------|---------|
| DC.B | '[TAB]' |
|------|---------|

new42ue:

new42ss:

| | |
|------|-----------------|
| DC.B | '[SHIFTED-TAB]' |
|------|-----------------|

new42se:

The new high map table points to the string descriptor at address newkey42. The new high map types table says that there is one qualifier, which means that there are two strings in the

key string descriptor.

Each string in the descriptor takes two bytes in this part of the table: the first byte is the length of the string, and the second byte is the distance from the start of the descriptor to the start of the string. Therefore, a single string (KCF_STRING + KC_NOQUAL) takes 2 bytes of string descriptor. If there is one qualifier, 4 bytes of descriptor are used. If there are two qualifiers, 8 bytes of descriptor are used. If there are 3 qualifiers, 16 bytes of descriptor are used. All strings start immediately following the string descriptor in that they are accessed as single-byte offsets from the start of the descriptor itself. Therefore, the distance from the start of the descriptor to the last string in the set (the one that uses the entire set of specified qualifiers) must start within 255 bytes of the descriptor address.

Because the length of the string is contained in a single byte, the length of any single string must be 255 bytes or less while also meeting the “reach” requirement. However, the console input buffer size limits the string output from any individual key to 32 bytes maximum.

The length of a keymap containing string descriptors and strings is variable and depends on the number and size of the strings that you provide.

CAPSABLE BIT TABLE

The vectors called `km_LoCapsable` and `km_HiCapsable` point to the first byte in an 8-byte table that contains more information about the keytable entries. Specifically, if the Caps Lock key has been pressed (the Caps Lock LED is on) and if there is a bit *on* in that position in the capsable map, then this key will be treated as though the Shift key is now currently pressed. For example, in the default key mapping, the alphabetic keys are “capsable” but the punctuation keys are not. This allows you to set the Caps Lock key, just as on a normal typewriter, and get all capital letters. However, unlike a normal typewriter, you need not go out of Caps Lock to correctly type the punctuation symbols or numeric keys.

In the table, the bits that control this feature are numbered from the lowest bit in the byte, and from the lowest memory byte address to the highest. For example, the bit representing capsable status for the key that transmits raw code 00 is bit 0 in byte 0; for the key that transmits raw code 08 it is bit 0 in byte 1, and so on.

There are 64 bits (8-bytes) in each of the two capsable tables.

REPEATABLE BIT TABLE

For both the low and high key maps there is an 8-byte table that provides one bit per possible raw key code. This bit indicates whether or not the specified key should repeat at the rate set by the Preferences program. The bit positions correspond to those specified in the capsable bit table.

If there is a 1 in a specific position, the key can repeat. The vectors that point to these tables are called `km_LoRepeatable` and `km_HiRepeatable`.

DEFAULT LOW KEY MAP

In the default low key map, all of the keys are treated in the same manner:

- o When pressed alone, they transmit the ASCII equivalent of the unshifted key.
- o When Shifted, they translate the ASCII equivalent of the shifted value when printed on the keycap.
- o When "Alt'ed" (pressed along with an Alt key), they transmit the alone-value with the high bit of a byte set (value plus hex 80).
- o When Shifted and Alt'ed, they transmit the shifted-value plus hex 80.

In this table, the bytes that describe the data to be transmitted are positioned as the example for the "A" key shown here:

| | | | |
|--------------------|-------------------|-------------------------|-------------------------------------|
| <code>key_A</code> | <code>DC.B</code> | <code>('A')+\$80</code> | <code>;Shifted and Alt'ed</code> |
| | <code>DC.B</code> | <code>('a')+\$80</code> | <code>;Alt'ed only</code> |
| | <code>DC.B</code> | <code>('A')</code> | <code>;Shifted only</code> |
| | <code>DC.B</code> | <code>('a')</code> | <code>;not Shifted or Alt'ed</code> |

In addition to the response to the key alone, Shifted, Alt'ed, and Shifted-and-Alt'ed, the default low keymap also responds to the key combination of "Ctrl + key" by stripping off bits 6 and 5 of the generated data byte. For example, Ctrl + A generates the translated keycode 01 (61 with bits 6 and 5 set to 0).

All keys in the low key map are mapped to their ASCII equivalents, as noted in the low key map key table shown above.

Because the low key table contains 4 bytes per key, and describes the keys (raw codes) from hex 00-3F, there are 64 times 4 or 256 bytes in this table.

DEFAULT HIGH KEY MAP

Most of the keys in the high key map generate strings rather than single-character mapping. The following keys map characters with no qualifier, along with their byte mapping:

| Key | Generates Value: |
|--------|------------------|
| BACKSP | \$08 |
| ENTER | \$0D |
| DEL | \$7F |

The following keys map characters and use a single qualifier:

| Key | Generates Value: | If Used with Qualifier,
Generates Value: |
|-----------------|------------------|---|
| SPACE | \$20 | \$A0 (qualifier = ALT) |
| RETURN | \$0D | \$0A (qualifier = CONTROL) |
| ESC | \$1B | \$9B (qualifier = ALT) |
| numeric pad “-” | \$2D | \$FF (qualifier = ALT) |

The following keys generate strings:

| Key | Generates Value: | If Used with <SHIFT>, generates Value: |
|----------------|------------------------|--|
| TAB | \$09 | \$9B, followed by 'Z' |
| cursor: | | |
| UP | \$9B, followed by 'A' | \$9B, followed by 'T' |
| DOWN | \$9B, followed by 'B' | \$9B, followed by 'S' |
| FWD | \$9B, followed by 'C' | \$9B, followed by ' ', followed by '@' |
| BACKWD | \$9B, followed by 'D' | \$9B, followed by ' ', followed by 'A' |
| function keys: | | |
| F1 | \$9B, followed by '0~' | \$9B, followed by '10~' |
| F2 | \$9B, followed by '1~' | \$9B, followed by '11~' |
| F3 | \$9B, followed by '2~' | \$9B, followed by '12~' |
| F4 | \$9B, followed by '3~' | \$9B, followed by '13~' |
| F5 | \$9B, followed by '4~' | \$9B, followed by '14~' |
| F6 | \$9B, followed by '5~' | \$9B, followed by '15~' |
| F7 | \$9B, followed by '6~' | \$9B, followed by '16~' |
| F8 | \$9B, followed by '7~' | \$9B, followed by '17~' |
| F9 | \$9B, followed by '8~' | \$9B, followed by '18~' |
| F10 | \$9B, followed by '9~' | \$9B, followed by '19~' |
| HELP | \$9B, followed by '?~' | (no qualifier used) |

Closing a Console Device

When you have finished using a console, it must be closed so that the memory areas it utilized may be returned to the system memory manager. Here is a sequence that you can use to close a console device:

```
CloseDevice(requestBlock);
```

Note that you should also delete the messages and ports associated with this console after the console has been closed:

```
DeleteStdIO(consoleWriteMsg);
DeleteStdIO(consoleReadMsg);
DeletePort(consoleWritePort);
DeletePort(consoleReadPort);
```

If you have finished with the window used for the console device, you can now close it.

Example Program

The following is a console device demonstration program with supporting macro routines.

```
/* cons.c */

/* This program is supported by the Amiga C compiler, version 1.1 and beyond.
 * (v1.0 compiler has difficulties if string variables do not have their initial
 * character aligned on a longword boundary. Compiles acceptably but won't run
 * correctly.)
 */

#include "exec/types.h"
#include "exec/io.h"
#include "exec/memory.h"

#include "graphics/gfx.h"
#include "hardware/dmabits.h"
#include "hardware/custom.h"
#include "hardware/blit.h"
#include "graphics/gfxmacros.h"
#include "graphics/copper.h"
#include "graphics/view.h"
#include "graphics/gels.h"
#include "graphics/regions.h"
#include "graphics/clip.h"
#include "exec/exec.h"
#include "graphics/text.h"
#include "graphics/gfxbase.h"

#include "devices/console.h"
#include "devices/keymap.h"

#include "libraries/dos.h"
```

```

#include "graphics/text.h"
#include "libraries/diskfont.h"
#include "intuition/intuition.h"

```

```

UBYTE escdata[] = { 0x9b, '@', /* insert character */
    0x9b, 'A', /* cursor up */
    0x9b, 'B', /* cursor down */
    0x9b, 'C', /* cursor left */
    0x9b, 'D', /* cursor right */
    0x9b, 'E', /* cursor next line */
    0x9b, 'F', /* cursor prev line */
    0x9b, 'J', /* erase to end of display */
    0x9b, 'K', /* erase to end of line */
    0x9b, 'L', /* insert line */
    0x9b, 'M', /* delete line */
    0x9b, 'P', /* delete character */
    0x9b, 'S', /* scroll up */
    0x9b, 'T', /* scroll down */
    0x1b, 'c', /* reset to initial state */
    0x9b, 'q', /* window status request */
    0x9b, 'n', /* device status report */
    0x9b, ' ', 'p', /* cursor on */
    0x9b, '0', ' ', 'p', /* cursor off */
    0x9b, '2', '0', 'h', /* set mode */
    0x9b, '2', '0', 'l', /* reset mode */
};

```

/* COVER A SELECTED SUBSET OF THE CONSOLE AVAILABLE FUNCTIONS */

```

#define INSERTCHARSTRING &escdata[0]
#define CURSUPSTRING &escdata[0+2]
#define CURSDOWNSTRING &escdata[0+4]
#define CURSFWDSTRING &escdata[0+6]
#define CURSBAKSTRING &escdata[0+8]
#define CURSNEXTLINE &escdata[0+10]
#define CURSPREVLIN &escdata[0+12]
#define ERASEEODSTRING &escdata[0+14]
#define ERASEEOLSTRING &escdata[0+16]
#define INSERTLINESTRING &escdata[0+18]
#define DELETELINESTRING &escdata[0+20]
#define DELCHARSTRING &escdata[0+22]
#define SCROLLUPSTRING &escdata[0+24]
#define SCROLLDOWNSTRING &escdata[0+26]
#define RESETINITSTRING &escdata[0+28]
#define WINDOWSTATSTRING &escdata[0+30]

```



```

#define DEVSTATSTRING      &escdata[0+32]
#define CURSONSTRING      &escdata[0+34]
#define CURSOFFSTRING     &escdata[0+37]
#define SETMODESTRING     &escdata[0+41]
#define RESETMODESTRING  &escdata[0+45]

#define BACKSPACE(r)      ConPutChar(r,0x08)
#define TAB(r)           ConPutChar(r,0x09)
#define LINEFEED(r)      ConPutChar(r,0x0a)
#define VERTICALTAB(r)   ConPutChar(r,0x0b)
#define FORMFEED(r)      ConPutChar(r,0x0c)
#define CR(r)            ConPutChar(r,0x0d)
#define SHIFTOUT(r)      ConPutChar(r,0x0e)
#define SHIFTIN(r)       ConPutChar(r,0x0f)
#define CLEARSCREEN(r)   ConPutChar(r,0x0c)

#define RESET(r)         ConWrite(r,RESETINITSTRING,2)
#define INSERT(r)        ConWrite(r,INSERTCHARSTRING,2)
#define CURSUP(r)        ConWrite(r,CURSUPSTRING,2)
#define CURSDOWN(r)      ConWrite(r,CURSDOWNSTRING,2)
#define CURSFWD(r)       ConWrite(r,CURSFWDSTRING,2)
#define CURSBAK(r)       ConWrite(r,CURSBAKSTRING,2)
#define CURSNEXTLN(r)    ConWrite(r,CURSNEXTLINE,2)
#define CURSPREVLN(r)    ConWrite(r,CURSPREVLIN,2)
#define ERASEEOD(r)      ConWrite(r,ERASEEODSTRING,2)
#define ERASEEOL(r)      ConWrite(r,ERASEEOLSTRING,2)
#define INSERTLINE(r)    ConWrite(r,INSERTLINESTRING,2)
#define DELETELINE(r)    ConWrite(r,DELETELINESTRING,2)
#define SCROLLUP(r)      ConWrite(r,SCROLLUPSTRING,2)
#define SCROLLDOWN(r)    ConWrite(r,SCROLLDOWNSTRING,2)
#define DEVICESTATUS(r)  ConWrite(r,DEVSTATSTRING,2)
#define WINDOWSTATUS(r) ConWrite(r,WINDOWSTATSTRING,2)
#define DELCHAR(r)       ConWrite(r,DELCHARSTRING,2)
#define CURSORON(r)      ConWrite(r,CURSONSTRING,3)
#define CURSOROFF(r)     ConWrite(r,CURSOFFSTRING,4)
#define SETMODE(r)       ConWrite(r,SETMODESTRING,4)
#define RESETMODE(r)     ConWrite(r,RESETMODESTRING,4)

#define CloseConsole(r)  CloseDevice(r)

ULONG DosBase;
ULONG DiskfontBase;
ULONG IntuitionBase;
ULONG GfxBase;

```

```

struct NewWindow nw = {
    10, 10,          /* starting position (left,top) */
    620,90,         /* width, height */
    -1,-1,         /* detailpen, blockpen */
    0,              /* flags for idcmp */
    WINDOWDEPTH|WINDOWSIZING|WINDOWDRAG|SIMPLE_REFRESH
    |ACTIVATE|GIMMEZEROZERO, /* window gadget flags */
    0,              /* pointer to 1st user gadget */
    NULL,           /* pointer to user check */
    "Console Test", /* title */
    NULL,           /* pointer to window screen */
    NULL,           /* pointer to super bitmap */
    100,45,         /* min width, height */
    640,200,        /* max width, height */
    WBENCHSCREEN};

```

```

struct Window *w;
struct RastPort *rp;

```

```

struct IOStdReq *consoleWriteMsg; /* I/O request block pointer */
struct MsgPort *consoleWritePort; /* a port at which to receive */
struct IOStdReq *consoleReadMsg; /* I/O request block pointer */
struct MsgPort *consoleReadPort; /* a port at which to receive */

```

```

extern struct MsgPort *CreatePort();
extern struct IOStdReq *CreateStdIO();

```

```

char readstring[200]; /* provides a buffer even though using only one char */

```

```

main()
{

```

```

    SHORT i;
    SHORT status;
    SHORT problem;
    SHORT error;
    problem = 0;

```

```

    if((DosBase = OpenLibrary("dos.library", 0)) == NULL)
        { problem = 1; goto cleanup1; }
    if((DiskfontBase=OpenLibrary("diskfont.library",0))==NULL)
        { problem = 2; goto cleanup2; }
    if((IntuitionBase=OpenLibrary("intuition.library",0))==NULL)
        { problem = 3; goto cleanup3; }
    if((GfxBase=OpenLibrary("graphics.library",0))==NULL)

```

```

    { problem = 4; goto cleanup4; }

consoleWritePort = CreatePort("my.con.write",0);
if(consoleWritePort == 0)
    { problem = 5; goto cleanup5; }
consoleWriteMsg = CreateStdIO(consoleWritePort);
if(consoleWritePort == 0)
    { problem = 6; goto cleanup6; }

consoleReadPort = CreatePort("my.con.read",0);
if(consoleReadPort == 0)
    { problem = 7; goto cleanup7; }
consoleReadMsg = CreateStdIO(consoleReadPort);
if(consoleReadPort == 0)
    { problem = 8; goto cleanup8; }

w = (struct Window *)OpenWindow(&nw); /* create a window */
if(w == NULL)
    { problem = 9; goto cleanup9; }

rp = w->RPort;          /* establish its rastport for later */

/* ***** */
/* NOW, Begin using the actual console macros defined above.          */
/* ***** */

error = OpenConsole(consoleWriteMsg,consoleReadMsg,w);
if(error != 0)
    { problem = 10; goto cleanup10; }
/* attach a console to this window, initialize
 * for both write and read */

QueueRead(consoleReadMsg,&readstring[0]); /* tell console where to
                                           * put a character that
                                           * it wants to give me
                                           * and queue up first read */
ConWrite(consoleWriteMsg,"Hello, World\r\n",14);

ConPutStr(consoleWriteMsg,"testing BACKSPACE");
for(i=0; i<10; i++)
    { BACKSPACE(consoleWriteMsg); Delay(30); }

ConPutStr(consoleWriteMsg,"\r\n");

ConPutStr(consoleWriteMsg,"testing TAB\r");

```

```

for(i=0; i<6; i++)
    { TAB(consoleWriteMsg); Delay(30); }

ConPutStr(consoleWriteMsg, "\r\n");

ConPutStr(consoleWriteMsg, "testing LINEFEED\r");
for(i=0; i<4; i++)
    { LINEFEED(consoleWriteMsg); Delay(30); }

ConPutStr(consoleWriteMsg, "\r\n");

ConPutStr(consoleWriteMsg, "testing VERTICALTAB\r");
for(i=0; i<4; i++)
    { VERTICALTAB(consoleWriteMsg); Delay(30); }

ConPutStr(consoleWriteMsg, "\r\n");

ConPutStr(consoleWriteMsg, "testing FORMFEED\r");
Delay(30);
for(i=0; i<2; i++)
    { FORMFEED(consoleWriteMsg); Delay(30); }

ConPutStr(consoleWriteMsg, "\r\n");

ConPutStr(consoleWriteMsg, "testing CR");
Delay(30);
CR(consoleWriteMsg);
Delay(60);
ConPutStr(consoleWriteMsg, "\r\n");

ConPutStr(consoleWriteMsg, "testing INSERT\r");
for(i=0; i<4; i++)
    { INSERT(consoleWriteMsg); Delay(30); }

ConPutStr(consoleWriteMsg, "\r\n");

ConPutStr(consoleWriteMsg, "testing DELCHAR\r");
CR(consoleWriteMsg);
for(i=0; i<4; i++)
    { DELCHAR(consoleWriteMsg); Delay(30); }

ConPutStr(consoleWriteMsg, "\r\n");

ConPutStr(consoleWriteMsg, "testing INSERTLINE\r");
CR(consoleWriteMsg);

```

```

for(i=0; i<3; i++)
    { INSERTLINE(consoleWriteMsg); Delay(30); }
ConPutStr(consoleWriteMsg, "\r\n");

ConPutStr(consoleWriteMsg, "testing DELETELINE\r");
CR(consoleWriteMsg);
LINEFEED(consoleWriteMsg);
Delay(60);
for(i=0; i<4; i++)
    { DELETELINE(consoleWriteMsg); Delay(30); }
ConPutStr(consoleWriteMsg, "\r\n");

ConPutStr(consoleWriteMsg, "testing CURSUP\r");
for(i=0; i<4; i++)
    { CURSUP(consoleWriteMsg); Delay(30); }

ConPutStr(consoleWriteMsg, "\r\n");

ConPutStr(consoleWriteMsg, "testing CURSDOWN\r");
for(i=0; i<4; i++)
    { CURSDOWN(consoleWriteMsg); Delay(30); }

ConPutStr(consoleWriteMsg, "\r\n");

ConPutStr(consoleWriteMsg, "testing CURSFWD\r");
for(i=0; i<4; i++)
    { CURSFWD(consoleWriteMsg); Delay(30); }

ConPutStr(consoleWriteMsg, "\r\n");

ConPutStr(consoleWriteMsg, "testing CURSBAK");
for(i=0; i<4; i++)
    { CURSBAK(consoleWriteMsg); Delay(30); }

ConPutStr(consoleWriteMsg, "\r\n");

ConPutStr(consoleWriteMsg, "testing CURSPREVLN");
for(i=0; i<4; i++)
    { CURSPREVLN(consoleWriteMsg); Delay(30); }

ConPutStr(consoleWriteMsg, "\r\n");

ConPutStr(consoleWriteMsg, "testing CURSNEXTLN");
for(i=0; i<4; i++)
    { CURSNEXTLN(consoleWriteMsg); Delay(30); }

```

```

ConPutStr(consoleWriteMsg, "\r\n");

ConPutStr(consoleWriteMsg, "testing ERASEEOD");
CURSPREVLN(consoleWriteMsg);
CURSPREVLN(consoleWriteMsg);
CURSPREVLN(consoleWriteMsg);
Delay(60);
for(i=0; i<4; i++)
    { ERASEEOD(consoleWriteMsg); Delay(30); }

ConPutStr(consoleWriteMsg, "\r\n");

ConPutStr(consoleWriteMsg, "testing ERASEEOL.junk");
CURSBAK(consoleWriteMsg);
CURSBAK(consoleWriteMsg);
CURSBAK(consoleWriteMsg);
CURSBAK(consoleWriteMsg);
CURSBAK(consoleWriteMsg);
Delay(60);
ERASEEOL(consoleWriteMsg);
Delay(30);
ConPutStr(consoleWriteMsg, "\r\n");

ConPutStr(consoleWriteMsg, "testing SCROLLUP");
for(i=0; i<4; i++)
    { SCROLLUP(consoleWriteMsg); Delay(30); }

ConPutStr(consoleWriteMsg, "\r\n");
ConPutStr(consoleWriteMsg, "testing SCROLLDOWN");
ConPutStr(consoleWriteMsg, "\n\n\n");
for(i=0; i<4; i++)
    { SCROLLDOWN(consoleWriteMsg); Delay(30); }

ConPutStr(consoleWriteMsg, "\r\n");

ConPutStr(consoleWriteMsg, "testing CURSOROFF");
CURSOROFF(consoleWriteMsg);
ConPutStr(consoleWriteMsg, "printed.with.cursor.off");
Delay(60);
ConPutStr(consoleWriteMsg, "\r\n");

CURSORON(consoleWriteMsg); Delay(30);
ConPutStr(consoleWriteMsg, "testing CURSORON");

```

```

/* ***** */
    Delay(120); /* wait 2 seconds (120/60 ticks) */

    status = CheckIO(consoleReadMsg); /* see if console read
                                       * anything, abort if not */
    if(status == FALSE) AbortIO(consoleReadMsg);
    WaitPort(consoleReadPort); /* wait for abort to complete */
    GetMsg(consoleReadPort); /* and strip message from port */

    CloseConsole(consoleWriteMsg);
cleanup10:
cleanup9:
    CloseWindow(w);
cleanup8:
    DeleteStdIO(consoleReadMsg);
cleanup7:
    DeletePort(consoleReadPort);
cleanup6:
    DeleteStdIO(consoleWriteMsg);
cleanup5:
    DeletePort(consoleWritePort);
cleanup4:
    CloseLibrary(GfxBase);
cleanup3:
    CloseLibrary(IntuitionBase);
cleanup2:
    CloseLibrary(DiskfontBase);
cleanup1:
    CloseLibrary(DosBase);
    if(problem > 0) exit(problem+1000);
    else
        return(0);
} /* end of main() */

/* Open a console device */

/* this function returns a value of 0 if the console
 * device opened correctly and a nonzero value (the error
 * returned from OpenDevice) if there was an error.
 */

int
OpenConsole(writerequest,readrequest,window)

```

```

struct IOStdReq *writerequest;
struct IOStdReq *readrequest;
struct Window *window;
{
    int error;
    writerequest->io_Data = (APTR) window;
    writerequest->io_Length = sizeof(*window);
    error = OpenDevice("console.device", 0, writerequest, 0);
    readrequest->io_Device = writerequest->io_Device;
    readrequest->io_Unit = writerequest->io_Unit;
    /* clone required parts of the request */
    return(error);
}

```

/* Output a single character to a specified console */

```

int
ConPutChar(request,character)
struct IOStdReq *request;
char character;
{
    request->io_Command = CMD_WRITE;
    request->io_Data = (APTR)&character;
    request->io_Length = 1;
    DoIO(request);
    /* command works because DoIO blocks until command is
     * done (otherwise pointer to the character could become
     * invalid in the meantime).
     */
    return(0);
}

```

/* Output a stream of known length to a console */

```

int
ConWrite(request,string,length)
struct IOStdReq *request;
char *string;
int length;
{
    request->io_Command = CMD_WRITE;
    request->io_Data = (APTR)string;
    request->io_Length = length;
    DoIO(request);
    /* command works because DoIO blocks until command is

```



```

        * done (otherwise pointer to string could become
        * invalid in the meantime).
        */
        return(0);
    }

/* Output a NULL-terminated string of characters to a console */

    int
ConPutStr(request,string)
    struct IOStdReq *request;
    char *string;
    {
        request->io_Command = CMD_WRITE;
        request->io_Data = (APTR)string;
        request->io_Length = -1; /* tells console to end when it sees
                                * a terminating zero on the string. */

        DoIO(request);
        return(0);
    }

/* queue up a read request to a console, show where to put the
 * character when ready to be returned. Most efficient if this is
 * called right after console is opened */

    int
QueueRead(request,wheret)
    struct IOStdReq *request;
    char *wheret;
    {
        request->io_Command = CMD_READ;
        request->io_Data = (APTR)wheret;
        request->io_Length = 1;
        SendIO(request);
        return(0);
    }

/* see if there is a character to read. If none, don't wait,
 * come back with a value of -1 */

    int
ConMayGetChar(request,requestPort, wheret)
    struct IOStdReq *request;
    char *wheret;
    {

```

```

register temp;

if ( GetMsg(requestPort) == NULL ) return(-1);
temp = *wheret;
QueueRead(request,wheret);
return(temp);
}

```

```

/* go and get a character; put the task to sleep if
 * there isn't one present */

```

UBYTE

```

ConGetChar(consolePort,request,wheret)
struct IOStdReq *request;
struct MsgPort *consolePort;
char *wheret;
{
register UBYTE temp;
while((GetMsg(consolePort) == NULL)) WaitPort(consolePort);
temp = *wheret; /* get the character */
QueueRead(request,wheret);
return(temp);
}

```


Chapter 9

Input Device

This chapter describes the Amiga input device, which is a combination of three other devices: keyboard device, gameport device, and timer device. The input device merges separate input event streams from the keyboard, mouse, and timer into a single stream. This single stream can then be interpreted by the prioritized linked list of input handlers that are watching the input stream.

Note that two additional messages can appear in the input stream: “disk inserted” and “disk removed.” These messages come from AmigaDOS and are sent to the input device for further propagation.

Introduction

The input device is automatically opened by AmigaDOS by any call to open the console device. When the input device is opened, a task, appropriately named "input.device", is started. The input device task communicates directly with the keyboard device to obtain raw key inputs. It also communicates with the gameport device to obtain mouse button and mouse movement events and with the timer device to obtain time events. In addition to these event streams, you can also directly input an event to the input device, to be fed to the handler chain. This topic is also covered below.

The keyboard device is also accessible directly (see chapter 10). However, while the input device task is operating, that task attempts to retrieve all incoming keyboard events and add them to the input stream.

The gameport device has two units. As you view the Amiga, looking at the gameport connectors, connector "1" is assigned as the primary mouse input for Intuition and contributes gameport input events to the input event stream. Connector "2" is handled by the other gameport unit and is currently unassigned. Each unit of the gameport device is an exclusive access object, in that you can specify what type of controller is attached. It is then assumed that only one task is sending requests for input from that unit. While the input device task is running, that task expects to read the input from connector 1. Direct use of the gameport device is covered in a separate chapter of this manual.

The timer device provides time events for the input device. It also provides time interval reports for controlling key repeat rate and key repeat threshold. The timer device is a shared-access device and is described in its own separate chapter.

Input Device Commands

The input device allows the following system functions:

| Command | Operation |
|----------------------|---|
| OpenDevice() | Obtain shared use of the input device |
| CloseDevice() | Relinquish use of the input device |
| DoIO() | Initiate a command, and wait for it to complete |
| SendIO() | Initiate a command, and return immediately |
| AbortIO() | Abort a command already in the queue |

Only the Start, Stop, Invalid, and Flush commands have been implemented for this device. All other commands are no-operations.

The input device also supports the device-specific commands shown in table 9-1.

Table 9-1: Input Device Commands

| I/O Command | Operation |
|----------------|--|
| IND_WRITEEVENT | Propagate an input event stream to all devices |
| IND_ADDHANDLER | Add an input-stream handler into the handler chain |
| IND_REMHANDLER | Remove an input-stream handler from the handler chain |
| IND_SETTHRESH | Set the repeating key hold-down time before repeat starts |
| IND_SETPERIOD | Set the period at which a repeating key repeats. |
| IND_SETMPORT | Set the gameport port to which the mouse is connected |
| IND_SETMTRIG | Read conditions that must be met by a mouse before
a pending read request will be satisfied |
| IND_SETMTYPE | Set the type of device at the mouse port |

The device-specific commands outlined above are described in the following paragraphs. A description of the contents of an input event is given first because the input device deals in input events. An input event is a data structure that describes the following:

- o The class of the event—often describes the device that generated the event
- o The subclass of the event—space for more information if needed
- o The code—keycode if keyboard, button information if mouse, others
- o A qualifier such as “Alt key also down,” “key repeat active”
- o A position field that contains a data address or a mouse position count
- o A time stamp, showing the sequence in which events have occurred
- o A link-field by which input events are linked together

The various types of input events are listed in the include file *devices/inpotevent.h*. That information is not repeated here. You can find more information about input events in the chapters titled “Gameport Device” and “Console Device.”

There is a difference between simply receiving an input event from a device (gameport, keyboard, or console) and actually becoming a handler of an input event stream. A handler is a routine that is passed an input event, and it is up to the handler to decide if it can process the

input event. If the handler does not recognize the event, it passes the address of the event as a return value.

Because of the input event field called `ie_NextEvent`, it is possible for the input event to be a pointer to the first event in a linked list of events to be handled. Thus, the handler should be designed to handle multiple events if such a link is used. Note that handlers can themselves generate new linked lists of events which can be passed down to lower priority handlers.

IND_ADDHANDLER COMMAND

You add a handler to the chain using the command `IND_ADDHANDLER`. Assuming that you have a properly initialized an `IOStdReq` block as a result of a call to `OpenDevice()` (for the input device), here is a typical C-language call to the `IND_ADDHANDLER` function:

```
struct Interrupt handlerStuff;
handlerStuff.is_Data = &hsData;
    /* address of its data area */
handlerStuff.is_Code = myhandler;
    /* address of entry point to handler */
handlerStuff.is_Node.In_Pri = 51;
    /* set the priority one step higher than Intuition, so that our
    * handler enters the chain ahead of Intuition.
    */
inputRequestBlock.io_Command = IND_ADDHANDLER;
inputRequestBlock.io_Data = &handlerStuff;

DoIO(&inputRequestBlock);
```

Notice from the above that Intuition is one of the input device handlers and normally distributes all of the input events. Intuition inserts itself at priority position 50. You can choose the position in the chain at which your handler will be inserted by setting the priority field in the list-node part of the interrupt data structure you are feeding to this routine.

Note also that *any* processing time expended by a handler subtracts from the time available before the next event happens. Therefore, handlers for the input stream must be fast.

Rules for Input Device Handlers

The following rules should be followed when you are designing an input handler:

- o If an input handler is capable of processing a specific kind of an input event and that event has no links (**ie_NextEvent** = 0), the handler can end the handler chain by returning a NULL (0) value.
- o If there are multiple events linked together, the handler is free to delink an event from the input event chain, thereby passing a shorter list of events to subsequent handlers. The starting address of the modified list is the return value.
- o If a handler wishes to add new events to the chain that it passes to a lower-priority handler, it may initialize memory to contain the new event or event chain. The handler, when it again gets control on the next round of event handling, should assume nothing about the current contents of the memory blocks it attached to the event chain. Lower priority handlers may have modified the memory as they handled their part of the event. The handler that allocates the memory for this purpose should keep track of the starting address and the size of this memory chunk so that the memory can be returned to the free memory list when it is no longer needed.

Your routine should be structured so that it can be called as though from the following C-language statement:

```
newEventChain = yourHandlerCode(oldEventChain, yourHandlerData);
```

where

- o **yourHandlerCode** is the entry point to your routine
- o **oldEventChain** is the starting address for the current chain of input events
- o **newEventChain** is the starting address of an event chain which you are passing to the next handler, if any

A NULL (0) value terminates the handling.

Memory that you use to describe a new input event that you have added to the event chain is available for reuse or deallocation when the handler is called again or after the **IND_REMHANDLER** command for the handler is complete.

Because **IND_ADDHANDLER** installs a handler in any position in the handler chain, it can, for example, ignore specific types of input events as well as act upon and modify existing streams of input. It can even create new input events for Intuition or other programs to interpret.

IND_REMHANDLER COMMAND

You remove a handler from the handler chain with the command `IND_REMHANDLER`. Assuming that you have a properly initialized `IOStdReq` block as a result of a call to `OpenDevice()` (for the input device) and you have already added the handler using `IND_ADDHANDLER`, here is a typical C-language call to the `IND_REMHANDLER` function:

```
inputRequestBlock.io_Command = IND_REMHANDLER;
inputRequestBlock.io_Data = &handlerStuff;
    /* tell it which one to remove */
DoIO(&inputRequestBlock);
```

IND_WRITEEVENT COMMAND

As noted in the overview of this chapter, input events are normally generated by the timer device, keyboard device or gameport device. A user can also generate an input event and send it to the input device. It will then be treated as any other event and passed through to the input handler chain. You can create your own stream of events and then send them to the input device using the `IND_WRITEEVENT` command. Here is an example, assuming a correctly initialized `input_request_block`. The example sends in a single event, which is a phony mouse-movement:

```
struct InputEvent phony;

input_request_block.io_Command = IND_WRITEEVENT;
input_request_block.io_Flags = 0;
input_request_block.io_Length = sizeof(struct InputEvent);
input_request_block.io_Data = &phony;

phony.ie_NextEvent = NULL;    /* only one */
phony.ie_Class = IECLASS_RAWMOUSE;
phony.ie_TimeStamp.tv_secs = 0;
phony.ie_TimeStamp.tv_micro = 0;
phony.ie_Code = IECODE_NOBUTTON;
phony.ie_Qualifier = IEQUALIFIER_RELATIVEMOUSE;
phony.ie_X = 10;
phony.ie_Y = 5;
    /* mouse didn't move, but program made system think that it did. */
DoIO(&input_request_block);
```

Note: This command adds the input event to the end of the current event stream. The system links other events onto the end of this event, thus modifying the contents of the data structure you constructed in the first place.

IND_SETTHRESH COMMAND

This command sets the timing in seconds and microseconds for the input device to indicate how long a user must hold down a key before it begins to repeat. This command is normally performed by the Preferences tool or by Intuition when it notices that the Preferences have been changed. If you wish, you can call this function. The following typical sequence assumes that you have already correctly initialized the request block by opening the input device. Only the fields shown here need be initialized.

```
struct InputEvent thresh_event;

input_request_block.io_Command = IND_SETTHRESH;
input_request_block.io_Flags = 0;
input_request_block.io_Data = &thresh_event;

thresh_event.ie_NextEvent = 0;
thresh_event.ie_TimeStamp.tv_secs = 1;    /* one second */
thresh_event.ie_TimeStamp.tv_micro = 500000;
/* 500,000 microseconds = 1/2 second */
DoIO(&input_request_block);
```

IND_SETPERIOD COMMAND

This command sets the time period between key repeat events once the initial period threshold has elapsed. Again, it is a command normally issued by Intuition and preset by the Preferences tool. A typical calling sequence is as shown above; change the command number and the timing period values to suit your application.

Input Device and Intuition

There are several ways to receive information from the various devices that are part of the input device. The first way is to communicate directly with the device. This way is, as specified above, occasionally undesirable (while the input device task is running). The second way is to become a handler for the stream of events which the input device produces. That method is also shown above.

The third method of getting input from the input device is to retrieve the data from the console device or from the IDCMP (Intuition Direct Communications Message Port).

If you choose this third method, you should be aware of what happens to input events if your task chooses not to respond to them. If there is no active window and no active console, then input events (keystrokes or left-button mouse clicks usually) will simply be ignored. If, however, there is an active window (yours), and you choose to simply let the messages pile up without responding to them as quickly as possible, here is what happens:

- o Another event occurs. If the system has no empty message that it can fill in to report this new event, then memory is dynamically allocated to hold this new information and the new message is transmitted to the message port for the task.
- o When the task finally responds to the message, the allocated memory is not returned to the system until the window is closed. Therefore, a task that chooses not to respond to its incoming messages for a long period of time can potentially remove a great deal of memory from the system free-memory list, making that memory space unavailable to this or other tasks until this task is completed.

Thus it is always a good idea to respond to input messages as quickly as possible to maximize the amount of free memory in the system while your task is running.

Sample Program

```
/* Sample program for adding an input handler to the input stream
 * Note that compiling this program native on the Amiga requires
 * a separate compile for this program, a separate assembly for the
 * handler.interface.asm, and a separate alink phase. Alink will
 * be used to tie together the object files produced by the separate
 * language phases. If compiling under Amiga C, disable stack checking
 * code in pass 2 of the compiler ( e.g., lc2 -v filename.q).
 *
 * Linking information:
 * inputdev.with:
 *
 * FROM lib:Lstartup.obj,inputdev.o, input.timerstuff.o, handler.interface.o
 * TO inputdev
 * LIBRARY lib:lc.lib, lib:amiga.lib
 */

#include <exec/types.h>
#include <exec/ports.h>
#include <exec/memory.h>
#include <exec/io.h>
#include <exec/tasks.h>
```

```

#include <exec/interrupts.h>
#include <devices/input.h>
#include <exec/devices.h>
#include <devices/inputevent.h>

#define F1KEYUP 0xD0
struct InputEvent copyevent; /* local copy of the event */
/* assumes never has a next.event attached */
struct MsgPort *inputDevPort;
struct IOStdReq *inputRequestBlock;
struct Interrupt handlerStuff;

struct InputEvent dummyEvent;

extern struct MsgPort *CreatePort();
extern struct IOStdReq *CreateStdIO();

struct MemEntry me[10];

/* If we want the input handler itself to add anything to the
 * input stream, we will have to keep track of any dynamically
 * allocated memory so that we can later return it to the system.
 * Other handlers can break any internal links the handler puts
 * in before it passes the input events.
 */

struct InputEvent
*myhandler(ev, mydata)
    struct InputEvent *ev; /* and a pointer to a list of events */
    struct MemEntry *mydata[];
/* system will pass me a pointer to my own data space. */
{
    /* Demo version of program simply reports input events as
     * its sees them; passes them on unchanged. Also, if there
     * is a linked chain of input events, reports only the lead
     * one in the chain, for simplicity.
     */
    if(ev->ie_Class == IECLASS_TIMER)
    {
        return(ev);
    }
    /* don't try to print timer events!!! they come every 1/10th sec. */
    else
    {
        Forbid(); /* don't allow a mix of events to be reported */
    }
}

```

```

    copyevent.ie_Class = ev->ie_Class;
    copyevent.ie_SubClass = ev->ie_SubClass;
    copyevent.ie_Code = ev->ie_Code;
    copyevent.ie_Qualifier = ev->ie_Qualifier;
    copyevent.ie_X = ev->ie_X;
    copyevent.ie_Y = ev->ie_Y;
    copyevent.ie_TimeStamp.tv_secs = ev->ie_TimeStamp.tv_secs;
    copyevent.ie_TimeStamp.tv_micro = ev->ie_TimeStamp.tv_micro;
    Permit();
}

/* There will be lots of events coming through here;
 * rather than make the system slow down because something
 * is busy printing the previous event, let's just print what
 * we find is current, and if we miss a few, so be it.
 *
 * Normally this loop would "handle" the event or perhaps
 * add a new one to the stream. (At this level, the only
 * events you should really be adding are mouse, rawkey or timer,
 * because you are ahead of the intuition interpreter.)
 * No printing is done in this loop (lets main() do it) because
 * printf can't be done by anything less than a 'process'
 */
return(ev);
/* pass on the pointer to the event (most handlers would
 * pass on a pointer to a changed or an unchanged stream)
 * (we are simply reporting what is seen, not trying to
 * modify it in any way) */
}

/* NOTICE: THIS PROGRAM LINKS ITSELF INTO THE INPUT STREAM AHEAD OF
 * INTUITION. THEREFORE THE ONLY INPUT EVENTS THAT IT WILL SEE AT
 * ALL ARE TIMER, KEYBOARD AND GAMEPORT. AS NOTED IN THE PROGRAM,
 * THE TIMER EVENTS ARE IGNORED DELIBERATELY */

extern struct Task *FindTask();
struct Task *mytask;
LONG mysignal;
extern VOID HandlerInterface();

struct timerequest *mytimerRequest;

extern struct timerequest *PrepareTimer();
extern int WaitTimer();
extern int DeleteTimer();

```

```

main()
{
    SHORT error;
    ULONG oldseconds, oldmicro, oldclass;

    /* init dummy event, this is what we will feed to other handlers
     * while this handler is active */

    dummyEvent.ie_Class = IECLASS_NULL; /* no event happened */
    dummyEvent.ie_NextEvent = NULL; /* only this one in the chain */

    inputDevPort = CreatePort(0,0); /* for input device */
    if(inputDevPort == NULL) exit(-1); /* error during createport */
    inputRequestBlock = CreateStdIO(inputDevPort);
    if(inputRequestBlock == 0) { DeletePort(inputDevPort); exit(-2); }
                                /* error during createstdio */

    mytimerRequest = PrepareTimer();
    if(mytimerRequest == NULL) exit(-3);

    handlerStuff.is_Data = (APTR)&me[0];
                                /* address of its data area */
    handlerStuff.is_Code = HandlerInterface;
                                /* address of entry point to handler */
    handlerStuff.is_Node.ln_Pri = 51;
                                /* set the priority one step higher than
     * Intuition, so that our handler enters
     * the chain ahead of Intuition.
     */

    error = OpenDevice("input.device",0,inputRequestBlock,0);
    if(error == 0) printf("\nOpened the input device");

    inputRequestBlock->io_Command = IND_ADDHANDLER;
    inputRequestBlock->io_Data = (APTR)&handlerStuff;

    DoIO(inputRequestBlock);
    copyevent.ie_TimeStamp.tv_secs = 0;
    copyevent.ie_TimeStamp.tv_micro = 0;
    copyevent.ie_Class = 0;
    oldseconds = 0;
    oldmicro = 0;
    oldclass = 0;

    for(;;) /* FOREVER */
    {

```

```

WaitForTimer(mytimerRequest, 0, 100000);
    /* TRUE = wait; time = 1/10th second */

/* note: while this task is asleep, it is very very likely that
 * one or more events will indeed pass through the input handler.
 * This task will only print a few of them, but won't intermix
 * the pieces of the input event itself because of the Forbid()
 * and Permit() (not allow task swapping when a data structure
 * isn't internally consistent)
 */
if(copyevent.ie_Class == IECLASS_RAWKEY && copyevent.ie_Code == F1KEYUP)
    break; /* exit from forever */
else
    {
        Forbid();
        if(copyevent.ie_TimeStamp.tv_secs != oldseconds ||
           copyevent.ie_TimeStamp.tv_micro != oldmicro ||
           copyevent.ie_Class != oldclass )
            {
                oldseconds = copyevent.ie_TimeStamp.tv_secs;
                oldmicro = copyevent.ie_TimeStamp.tv_micro;
                oldclass = copyevent.ie_Class;
                showEvents(&copyevent);
            }
        Permit();
    }
}
/* Although this task sleeps (main loop), the handler is independently
 * called by the input device.
 */

/* For keystrokes that might be recognized by AmigaDOS, such as
 * alphabetic or numeric keys, you will notice that after the
 * first such keystroke, AmigaDOS appears to lock out your task
 * and accepts all legal keystrokes until you finally hit return.
 * This is absolutely true.... when both you and AmigaDOS try to
 * write into the same window, as is true if you run this program
 * from the CLI, the first keystroke recognized by AmigaDOS locks
 * the layer into which it is writing. Any other task trying
 * to write into this same layer is put to sleep. This allows
 * AmigaDOS to edit the input line and prevents other output to
 * that same window from upsetting the input line appearance.
 * In the same manner, while your task is sending a line of output,
 * AmigaDOS can be put to sleep it too must output at that time.
 */

```

```

* You can avoid this problem if you wish by opening up a separate
* window and a console device attached to that window, and output
* strings to that console. If you click the selection button on
* this new window, then AmigaDOS won't see the input and your
* task will get to see all of the keystrokes. The other alternative
* you can use, for demonstration sake, is to:

```

- ```

*
* 1. Make the AmigaDOS window slightly smaller in the
* vertical direction.
* 2. Then click in the Workbench screen area outside
* of any window.

```

```

* Now there is no console device (particularly not AmigaDOS's
* console) receiving the raw key stream and your task will report
* as many keystrokes as it can catch (while not sleeping, that is).
*/

```

```

/* remove the handler from the chain */
inputRequestBlock->io_Command = IND_REMHANDLER;
inputRequestBlock->io_Data = (APTR)&handlerStuff;
DoIO(inputRequestBlock);

```

```

/* close the input device */
CloseDevice(inputRequestBlock);

```

```

/* delete the IO request */
DeleteStdIO(inputRequestBlock);

```

```

/* free other system stuff */
DeletePort(inputDevPort);
DeleteTimer(mytimerRequest);

```

```

} /* end of main */

```

```
int
```

```
showEvents(e)
```

```
struct InputEvent *e;
```

```
{
```

```

printf("\n\nNew Input Event");
printf("\nie_Class = %lx", e->ie_Class);
printf("\nie_SubClass = %lx", e->ie_SubClass);
printf("\nie_Code = %lx", e->ie_Code);
printf("\nie_Qualifier = %lx", e->ie_Qualifier);
printf("\nie_X = %ld", e->ie_X);
printf("\nie_Y = %ld", e->ie_Y);
printf("\nie_TimeStamp(seconds) = %lx", e->ie_TimeStamp.tv_secs);

```



```

 return(0);
}

/* input.timerstuff.c */

#include "exec/types.h"
#include "exec/nodes.h"
#include "exec/lists.h"
#include "exec/memory.h"
#include "exec/interrupts.h"
#include "exec/ports.h"
#include "exec/libraries.h"
#include "exec/io.h"
#include "exec/tasks.h"
#include "exec/execbase.h"
#include "exec/devices.h"
#include "devices/timer.h"

extern struct MsgPort *CreatePort();
extern struct IORequest *CreateExtIO();

struct timerequest
*PrepareTimer(precision)
SHORT precision;
{
 /* return a pointer to a time request. If any problem, return NULL */

 int error;
 SHORT whichunit;

 struct MsgPort *timerport;
 struct timerequest *timermsg;

 timerport = CreatePort(0,0);
 if (timerport == NULL)
 return(NULL); /* error during CreatePort */

 timermsg = (struct timerequest *)
 CreateExtIO(timerport,sizeof(struct timerequest));
 if (timermsg == NULL)
 {
 DeletePort(timerport);
 return(NULL); /* error during CreateExtIO */
 }
}

```

```

if(precision) /* if true, use precision timer (under 1 second) */
 whichunit = UNIT_MICROHZ;
else
 whichunit = UNIT_VBLANK;

error = OpenDevice(TIMERNAME, whichunit, timermsg, 0);
if (error != 0)
 {
 DeleteExtIO(timermsg,sizeof(struct timerequest));
 DeletePort(timerport);
 return(NULL); /* Error during OpenDevice */
 }
return(timermsg);
}

int
WaitForTimer(tr,seconds,microseconds)
ULONG seconds,microseconds;
struct timerequest *tr;
{
 tr->tr_node.io_Command = TR_ADDREQUEST; /* add a new timer request */
 tr->tr_time.tv_secs = seconds; /* seconds */
 tr->tr_time.tv_micro = microseconds; /* microseconds */
 DoIO(tr); /* post request to the timer */
 /* goes to sleep till done */
 return(0);
}

int
DeleteTimer(tr)
struct timerequest *tr;
{
 struct MsgPort *tp;

 tp = tr->tr_node.io_Message.mn_ReplyPort;
 if(tr != 0)
 {
 CloseDevice(tr);
 DeleteExtIO(tr,sizeof(struct timerequest));
 }
 if(tp != 0)
 DeletePort(tp);
 return(0);
}

```

\* handler.interface.asm

\*\*\*\*\*

\* HandlerInterface()

\*

\* This code is needed to convert the calling sequence performed by  
\* the input.task for the input stream management into something  
\* that a C program can understand.

\*

\* This routine expects a pointer to an InputEvent in A0, a pointer  
\* to a data area in A1. These values are transferred to the stack  
\* in the order that a C program would need to find them. Since the  
\* actual handler is written in C, this works out fine.

XREF \_myhandler

XDEF \_HandlerInterface

\_HandlerInterface:

MOVEM.L A0/A1,-(A7) ; save registers

JSR \_myhandler ; go to the C language routine we provided

ADDQ.L #8,A7 ; restore the registers on the way out.

RTS

END

# Chapter 10

## Keyboard Device

### Introduction

The keyboard device gives system access to the Amiga keyboard. When you send this device the command to read one or more keystrokes from the keyboard, for each keystroke (whether key-up or key-down) the keyboard device creates a data structure called an input event to describe what happened. A keyboard input event includes the key code (including up or down transition status), information about the current state of the left and right Shift keys, and whether the key came from the numeric keypad area.

Thus, the keyboard device provides more information than simply the “raw” key input that might be obtained by directly reading the hardware registers. In addition, the keyboard device can buffer keystrokes for you. If your task takes more time to process prior keystrokes, the keyboard device senses additional keystrokes and saves several keystrokes as a type-ahead feature. If your task takes an exceptionally long time to read this information from the keyboard, any keystrokes queued up beyond the number the system can handle will be ignored. Normally, the input device task processes these keyboard events, turning them into input device events so that no keystrokes are lost. You can find more information about keyboard event-queuing in the chapter, “Input Device,” in the topic titled “Input Device and Intuition.”

## Keyboard Device Commands

The keyboard device allows the following system functions. The system functions operate normally.

| Command              | Operation                                       |
|----------------------|-------------------------------------------------|
| <b>OpenDevice()</b>  | Obtain shared use of the keyboard device        |
| <b>CloseDevice()</b> | Relinquish use of the keyboard device           |
| <b>DoIO()</b>        | Initiate a command, and wait for it to complete |
| <b>SendIO()</b>      | Initiate a command, and return immediately      |
| <b>AbortIO()</b>     | Abort a command already in the queue            |

The keyboard device also responds to the following commands:

| I/O Command                 | Operation                                                                        |
|-----------------------------|----------------------------------------------------------------------------------|
| <b>KBD_ADDRESETHANDLER</b>  | Add a reset handler to the device                                                |
| <b>KBD_REMRESETHANDLER</b>  | Remove a reset handler from the device                                           |
| <b>KBD_RESETHANDLERDONE</b> | Indicate that a handler has completed its job and reset could possibly occur now |
| <b>KBD_READMATRIX</b>       | Read the state of every key in the keyboard                                      |
| <b>KBD_READEVENT</b>        | Read one (or more) key event from the keyboard device                            |

## KBD\_ADDRESETHANDLER

This command adds a routine to a chain of reset-handlers. When a user presses the key sequence Ctrl-left Amiga-right Amiga (the reset sequence), the keyboard device senses this and calls a prioritized chain of reset-handlers. These might be thought of as clean-up routines that “must” be performed before reset is allowed to occur. For example, if a disk write is in progress, the system should finish that before resetting the hardware so as not to corrupt the contents of the disk. There are probably a few reasons why a program may wish to add its own reset handler as well. Note that if you add your own handler to this chain, you *must* ensure that your handler allows the rest of reset processing to occur. Reset *must* continue to function.

You add a handler to the chain by the command KBD\_ADDRESETHANDLER. Assuming that you have a properly initialized **IOStdReq** block as a result of a call to **OpenDevice()** (for the input device), here is a typical C-language call to the KBD\_ADDRESETHANDLER function:

```
struct Interrupt resetHandlerStuff;
resetHandlerStuff.is_Data = &resetHandlerData;
 /* address of its data area */
resetHandlerStuff.is_Code = myResetHandler;
 /* address of entry point to handler */
resetHandlerStuff.is_Node.In_Pri = myPriority;
keyboardRequestBlock.io_Command = KBD_ADDRESETHANDLER;
keyboardRequestBlock.io_Data = &resetHandlerStuff;

DoIO(&keyboardRequestBlock);
```

The priority field in the list node structure establishes the sequence in which reset handlers are processed by the system. Your routine should be structured so that it can be called as though from the following C-language sequence:

```
myResetHandler(resetHandlerData);
```

Any return value from this routine is ignored. All keyboard reset handlers are activated if time permits.

The final command in your handler routine should be KBD\_RESETHANDLERDONE, as described below.

*Note:* Because of the time-critical nature of handlers, handlers are usually written in assembly code. However, keyboard reset processing can take a little longer and is therefore less critical if written in a language such as C.

## KBD\_REMRESETHANDLER

This command is used to remove a keyboard reset handler from the system. The only difference from the calling sequence shown in KBD\_ADDRESETHANDLER above is a change in the command number to KBD\_REMRESETHANDLER, and there is no need to specify the priority of the handler.

## KBD\_RESETHANDLERDONE

This command tells the system that this handler is finished with its essential activities. If this is the last handler in the chain, it completes the reset sequence. If not, the next handler in the chain gets its chance to function.

Here is a typical statement sequence used to end a keyboard reset handler, again assuming a properly initialized `inputRequestBlock`:

```
keyboardRequestBlock.io_Command = KBD_RESETHANDLERDONE;
keyboardRequestBlock.io_Data = &resetHandlerStuff;
SendIO(&keyboardRequestBlock);
return; /* return so that other handlers can also do their jobs */
```

Note that `SendIO()` is used instead of `DoIO()`. This routine is being executed within a software interrupt, and it is illegal to allow a `Wait()` within such routines.

## KBD\_READMATRIX

This command lets you discover the current state (UP = 0, DOWN = 1) of every key in the key matrix. You provide a data area that is at least large enough to hold one bit per key, approximately 16 bytes. The keyboard layout is shown in figure 10-1 below, indicating the numeric value each transmits (raw) when it is pressed. This value is the numeric position that this key occupies in the key matrix read by this command.

|             |                    |          |          |          |          |          |          |          |          |           |           |             |              |                     |             |         |         |
|-------------|--------------------|----------|----------|----------|----------|----------|----------|----------|----------|-----------|-----------|-------------|--------------|---------------------|-------------|---------|---------|
| ESC<br>45   | F1<br>50           | F2<br>51 | F3<br>52 | F4<br>53 | F5<br>54 | F6<br>55 | F7<br>56 | F8<br>57 | F9<br>58 | F10<br>59 | DEL<br>46 |             |              |                     |             |         |         |
| ~<br>00     | 1<br>01            | 2<br>02  | 3<br>03  | 4<br>04  | 5<br>05  | 6<br>06  | 7<br>07  | 8<br>08  | 9<br>09  | 0<br>0A   | -<br>0B   | =<br>0C     | OD           | BACK<br>SPACE<br>41 | 7<br>3D     | 8<br>3E | 9<br>3F |
| TAB<br>42   | Q<br>10            | W<br>11  | E<br>12  | R<br>13  | T<br>14  | Y<br>15  | U<br>16  | I<br>17  | O<br>18  | P<br>19   | [<br>1A   | ]<br>1B     | 44           | HELP<br>5F          | 4<br>2D     | 5<br>2E | 6<br>2F |
| CTRL<br>63  | CAPS<br>LOCK<br>62 | A<br>20  | S<br>21  | D<br>22  | F<br>23  | G<br>24  | H<br>25  | J<br>26  | K<br>27  | L<br>28   | ;"<br>29  | '"<br>2A    | RETURN<br>2B | ↑<br>4C             | 1<br>1D     | 2<br>1E | 3<br>1F |
| SHIFT<br>60 |                    | Z<br>30  | X<br>31  | C<br>32  | V<br>33  | B<br>34  | N<br>35  | M<br>36  | <<br>37  | ><br>38   | ?,<br>39  | SHIFT<br>3A |              | 61                  | ←<br>4F     | →<br>4E | 0<br>OF |
| ALT<br>64   | Alt<br>66          | 40       |          |          |          |          |          |          |          |           |           | Alt<br>67   | ALT<br>65    | ↓<br>4D             | ENTER<br>4A | 43      |         |

Figure 10-1: Raw Key Matrix

Assuming that you have already initialized an `IOStdReq` block for communication with the keyboard device, here is a typical calling sequence for sending the read-matrix command:

```

UBYTE keyMatrix[16];
keyboardRequestBlock.io_Command = KBD_READMATRIX;
keyboardRequestBlock.io_Data = &keyMatrix[0];
/* where to put the key matrix */
DoIO(&keyboardRequestBlock);

```

To find the status of a particular key (for example, to find out if the F2 key is down), you find the bit that specifies the current state by dividing the key matrix value (hex 51 = decimal 81) by 8. This indicates that the bit is in byte number 10 of the matrix. Then take the same number (decimal 81) modulo 8 to determine which bit position within that byte represents the state of the key. This yields a value of 1. So, by reading bit position 1 of byte number 10, you determine the status of the function key F2.

## KBD\_READEVENT

Reading keyboard events is normally not done through direct access to the keyboard device. See chapter 9, “Input Device,” for the intimate linkage between that device and the keyboard device. This section is provided primarily to show you the component parts of a keyboard input event.

The figure above shows the code value that each key places into the `ie_Code` field of the input event for a key down event. For a key-up event, a value of hexadecimal 80 is or’ed with the value shown above. Additionally, if either shift key is down, or if the key is one of those in the



numeric keypad, the qualifier field of the keyboard input event will be filled in accordingly.

*Note:* The keyboard device can queue up several keystrokes without a task requesting a report of keyboard events. However, when the keyboard event buffer has been filled with no task interaction, additional keystrokes will be discarded.

## Example Keyboard Read-event Program

*Note:* This sample program will run properly only if AmigaDOS and the input device are not active.

```
/* sample program to demonstrate direct communications with the keyboard,
 * won't work unless input device is disabled, so that keyboard can
 * be accessed individually. (It will compile and it will run, but
 * this program will get some of the keyboard's inputs, and the input
 * device will steal the rest... no guarantee that F1 Key can break it out.)
 *
 * To try the program, if run under the AmigaDOS CLI, strike any key, then
 * hit return. (You won't see any responses until each return key... DOS
 * is sitting on the input stream with its input editor as well as the
 * input device.) By rapidly hitting F1 then Return several times,
 * eventually you can generate a hex 50 that exits the program. This
 * program is provided for those who are taking over the machine. It
 * is not intended as a general purpose keyboard interface under DOS.
 */

#include <exec/types.h>
#include <exec/io.h>
#include <exec/devices.h>
#include <devices/keyboard.h>
#include <devices/inputevent.h>

#define F1KEY 0x50

extern struct MsgPort *CreatePort();
extern struct IOStdReq *CreateStdIO();

SHORT error;

struct IOStdReq *keyreq;
struct MsgPort *keyport;
struct InputEvent *keydata; /* pointer into the returned data area
 * where an input event has been sent */
```

```

BYTE keybuffer[sizeof(struct InputEvent)];

main()
{
 keyport = CreatePort(0,0);
 if(keyport == 0) { printf("\nError during CreatePort");
 exit(-1);
 }
 keyreq = CreateStdIO(keyport);
 /* make an io request block for
 * communicating with the keyboard */
 if(keyreq == 0) { printf("\nError during CreateStdIO");
 DeletePort(keyport);
 exit(-2);
 }
 error = OpenDevice("keyboard.device",0,keyreq,0);
 /* open the device for access */

 if (error != 0) { printf("\nCan't open keyboard!");
 ReturnMemoryToSystem();
 exit(-100);
 }

 keyreq->io_Length = sizeof(struct InputEvent);
 /* read one event each time we go back to the keyboard */

 keyreq->io_Data = (APTR)keybuffer;
 /* show where to put the data when read */

 keydata = (struct InputEvent *)keybuffer;

 keyreq->io_Command = KBD_READEVENT; /* get an event!! */

 for(;;) /* FOREVER */
 {
 printf("\n Ready to retrieve another key0);
 DoIO(keyreq);
 if(keydata->ie_Code == F1KEY) break;
 printf("\n Raw key found this time was %lx",keydata->ie_Code);
 }
 printf("\nFINALLY found an F1 key!!! Exiting...");
 ReturnMemoryToSystem(); /* can't get here because of FOREVER,
 * but if user provides an exit..... */
 }
}

```

```
ReturnMemoryToSystem()
{
 DeleteStdIO(keyreq);
 DeletePort(keyport);
 return(0);
}
```

# Chapter 11

## Gameport Device

### Introduction

The gameport device is the means of access to the Amiga gameports. There are two units in the gameport device. Unit 0 controls the front gameport connector (connector 1). Unit 1 controls the rear gameport connector (connector 2).

You must tell the system the type of device connected to the gameport connector and how the device is to respond. That is, should the device return status immediately each time you ask for information or should it only return status once certain conditions have been met?

When the input device is operating, the left gameport connector is usually dedicated to that device. Therefore, this chapter's examples concentrate on the right connector, which is not dedicated to the input device. Note that if the input device is not started, the left connector, as gameport unit 0, can perform the same functions as shown below for the right connector.

When a gameport unit finally responds to a request for input, it formulates an input event. The contents of the input event vary based on the type of device you have told the unit is connected and the trigger conditions it must look for.

## Gameport Device Commands

The gameport device allows the following system functions.

| Command              | Operation                                                                                                                                                 |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>OpenDevice()</b>  | Obtain exclusive use of one unit of the gameport device. Returns an error value of -1 if another task already has control of the unit you have requested. |
| <b>CloseDevice()</b> | Relinquish use of the gameport device                                                                                                                     |
| <b>DoIO()</b>        | Initiate a command and wait for it to complete                                                                                                            |
| <b>SendIO()</b>      | Initiate a command and return immediately                                                                                                                 |
| <b>AbortIO()</b>     | Abort a command already in the queue                                                                                                                      |

The gameport device also responds to the following commands:

| I/O Command           | Operation                                                   |
|-----------------------|-------------------------------------------------------------|
| <b>GPD_SETCTYPE</b>   | Set the type of the controller to be monitored              |
| <b>GPD_ASKCTYPE</b>   | Ask the type of the controller being monitored              |
| <b>GPD_SETTRIGGER</b> | Preset the conditions that will trigger a gameport event    |
| <b>GPD_ASKTRIGGER</b> | Inquire the conditions that have been preset for triggering |
| <b>GPD_READEVENT</b>  | Read one or more gameport events from an initialized unit   |

### **GPD\_SETCTYPE**

This command establishes the type of controller that is to be connected to the specific gameport device. You must have already successfully opened that specific unit before you will be able to tell it what type of controller is connected. As of this writing, there are three different legal controller types: mouse, absolute joystick, relative joystick, and "no controller."

A mouse controller can report input events for one, two, or three buttons and for positive or negative (x,y) movements. A trackball controller or driving controller for various games is generally of the same type, and can be declared as a mouse controller.

An absolute joystick is one that reports one single event for each change in its current location. If, for example, the joystick is centered and a user pushes the stick forward, a forward-switch event will be generated. A relative joystick, on the other hand, is comparable to an absolute joystick with "autorepeat" installed. As long as the user holds the stick in a position other than centered, the gameport device continues to generate position reports.

As of this writing, there is no direct system software support for proportional joysticks or proportional controllers.

You specify the controller type by the following code or its equivalent:

```
struct IOStdReq *gameIOMsg;

setControllerType(type)
 UBYTE *type;
 {
 /* set type of controller */
 gameIOMsg->io_Command = GPD_SETCTYPE;
 gameIOMsg->io_Data = type; /* show where data can be found */
 DoIO(gameIOMsg);
 return(0);
 }
```

## GPD\_GETCTYPE

You use this command to find out what kind of controller has been specified for a particular unit. This command puts the controller type into the data area that you specify with the command. Here is a sample call:

```
SHORT getControllerType(type);
 UBYTE *type;
 {
 /* get type of controller */
 gameIOMsg->io_Command = GPD_GETCTYPE;
 gameIOMsg->io_Data = type; /* show where data should be placed */
 DoIO(gameIOMsg);
 return (gamebuffer[0]);
 }
```

The value that is returned corresponds to one of the four controller types noted in GPD\_SETCTYPE above. Controller type definitions can be found in the include file named *devices/gameport.h*.

## GPD\_SETTRIGGER

You use this command to specify the conditions that can trigger a gameport event. The device won't reply to your read request until the trigger conditions have been satisfied.

For a mouse device, you can trigger on a certain minimum-sized move in either the x or y direction, on up or down transitions of the mouse buttons, on a timed basis, or any combination of these conditions. Here is an example that shows why you might want to use both time and movement. Suppose you normally signal mouse events if the mouse moves at least 10 counts in either the x or y directions. If you are moving the cursor to keep up with mouse movements and the user moves the mouse less than 10 counts, after a period of time you will want to update the position of the cursor to exactly match the mouse position. Thus the timed report with current mouse counts will be desirable.

For a joystick device, you can select timed reports as well as button-up and button-down report trigger conditions.

The information needed for gameport trigger setting is placed into a **GameTrigger** data structure:

```
struct GamePortTrigger {
 UWORD gpt_Keys; /* key transition triggers */
 UWORD gpt_Timeout; /* time trigger (vertical blank units) */
 UWORD gpt_XDelta; /* X distance trigger */
 UWORD gpt_YDelta; /* Y distance trigger */
};
```

The field **gpt\_Keys** can be set to a value of GPTF\_UPKEYS to report upward transitions or GPTF\_DOWNKEYS to report downward transitions.

The field **gpt\_Timeout** is set to count how many vertical blank units should occur (1/60th of a second each) between reports in the absence of another trigger condition. Thus, this specifies the maximum report interval.

*Note:* If a task sets trigger conditions and does not ask for the position reports (by sending an I/O request to be filled in with available reports), the gameport device will queue up several additional reports. If the trigger conditions again occur and as many events as the system can handle are already queued, the additional triggers will be ignored until the buffer of one or more of the existing triggers is read by a device read request.

```

struct GamePortTrigger mousetrigger = {
 GPTF_UPKEYS + GPTF_DOWNKEYS,
 1800,
 XMOVE,
 YMOVE };
 /* trigger on all mouse key transitions, every 30 seconds,
 * (1800 = 30 times 60 per sec) for any 10 in an x or y direction */

```

You set the trigger by using the following code or the equivalent:

```

gameIOMsg->io_Command = GPD_SETTRIGGER;
 /* command to set the trigger conditions */
gameIOMsg->io_Data = &mousetrigger;
 /* show where to find the trigger condition info */
DoIO(gameIOMsg);

```

## Example Programs

### MOUSE PROGRAM

Here is a complete sample program that lets you open the right gameport device unit and define it as a mouse device. You are directed to unplug the mouse and plug it into the right connector. Mouse moves and button clicks are reported to the console device that started the program. If you do not move the mouse for 30 seconds, a report is generated automatically. If you do not move it for 2 minutes, the program exits.

```

/* *****
* mouse test, for right game port on the Amiga
*
* Notes: The right port is used for this test because the input.device task is
* busy continuously with the lefthand port, feeding input events to Intuition or
* console devices. If Intuition is not activated (applications that take over the
* whole machine may decide not to activate Intuition) and if no console device is
* activated, * the input device will never activate, allowing the application free
* rein to use either the left OR the right hand joystick/mouse port. If either
* Intuition or the console device is activated, the lefthand port will yield, at
* best, every alternate input event to an external application such as this test program.
*
* This will undoubtedly mess up either of the two applications and should,

```



```

* therefore, be avoided. It was ok to use the right port in this case, because
* the system has no particular interest in monitoring it.
*
* Using a function called SetMPort(), you can reconfigure so that the
* mouse is expected in the other port, but that isn't demonstrated here.
***** */

```

```

#include <exec/types.h>
#include <exec/devices.h>
#include <graphics/gfx.h>
#include <devices/gameport.h>
#include <devices/inputevent.h>

```

```

LONG GfxBase=0;

```

```

#define XMOVE 10
#define YMOVE 10
#define MAX(m,n) (m > n ? m : n)

```

```

/* trigger on all mouse key transitions, and every
* 30 seconds, and for any 10 in an x or y direction */

```

```

struct GamePortTrigger mousetrigger = {
 GPTF_UPKEYS + GPTF_DOWNKEYS,
 1800,
 XMOVE,
 YMOVE };

```

```

struct InputEvent *game_data; /* pointer into the returned data area
* where input event has been sent */

```

```

SHORT error;

```

```

struct IOStdReq *game_io_msg;

```

```

BYTE gamebuffer[sizeof(struct InputEvent)];

```

```

BYTE *gamedata;

```

```

SHORT testval;

```

```

struct MsgPort *game_msg_port;

```

```

SHORT movesize;

```

```

extern struct MsgPort *CreatePort();

```

```

extern struct IOStdReq *CreateStdIO();

```

SHORT codeval, timeouts;

```
#define IF_NOT_IDLE_TWO_MINUTES while(timeouts < 4)
```

```
main()
```

```
{
```

```
 GfxBase = OpenLibrary("graphics.library", 0);
```

```
 if (GfxBase == NULL)
```

```
 {
```

```
 printf("Unable to open graphics library\n");
```

```
 exit(1000);
```

```
 }
```

```
 printf("Mouseport Demo\n");
```

```
 printf("\nMove Mouse from Left Port to Right Port\n");
```

```
 printf("\nThen move the mouse and click its buttons");
```

```
 timeouts = 0;
```

```
 gamedata = &gamebuffer[0];
```

```
 /* point to first location in game buffer */
```

```
 game_msg_port = CreatePort(0,0);
```

```
 /* provide a port for the IO response */
```

```
 if(game_msg_port == 0)
```

```
 {
```

```
 printf("\nError While Performing CreatePort");
```

```
 exit(-1);
```

```
 }
```

```
 game_io_msg = CreateStdIO(game_msg_port);
```

```
 /* make an io request block for communicating with
 the keyboard */
```

```
 if(game_io_msg == 0)
```

```
 {
```

```
 printf("\nError While Performing CreateStdIO");
```

```
 DeletePort(game_msg_port);
```

```
 exit(-2);
```

```
 }
```

```
 error = OpenDevice("gameport.device",1,game_io_msg,0);
```

```
 /* open the device for access, unit 1 is right port */
```

```
 if(error != 0)
```

```

{
 printf("\nError while opening the device, exiting");
 DeleteStdIO(game_io_msg);
 DeletePort(game_msg_port);
 exit(-3);
}

game_io_msg->io_Length = sizeof(struct InputEvent);
/* read one event each time we go back to the gameport */

game_io_msg->io_Data = (APTR)gamebuffer;
/* show where to put the data when read */

game_data = (struct InputEvent *)gamebuffer;

/* test the mouse in this loop */
set_controller_type(GPCT_MOUSE);

/* specify the trigger conditions */
game_io_msg->io_Command = GPD_SETTRIGGER;
/* show where to find the trigger condition info */
game_io_msg->io_Data = (APTR)&mousetrigger;

/* this command doesn't wait... returns immediately */
SendIO(game_io_msg);
WaitPort(game_msg_port);
GetMsg(game_msg_port);

printf("\nI will report:");
printf("\n Mouse X or Y moves if either is over 10 counts");
printf("\n Button presses (along with mouse moves if any)");
printf("\n Or every 30 seconds (along with mouse moves if any)");
printf("\n if neither move or click happens\n");
printf("\nIf no activity for 2 minutes, the program exits\n");

/* from now on, just read input events into the input buffer, one at a
*time. read-event waits for the preset conditions */

game_io_msg->io_Command = GPD_READEVENT;
game_io_msg->io_Data = (APTR)gamebuffer;

IF_NOT_IDLE_TWO_MINUTES
{
 game_io_msg->io_Length = sizeof(struct InputEvent);
 /* read one event each time we go back to the gameport */

```

```

printf("\n Waiting For Mouse Report\n");

SendIO(game_io_msg);

WaitPort(game_msg_port);
/* this is NOT a busy wait... it is a task-sleep */

GetMsg(game_msg_port);

codeval = game_data->ie_Code;
switch(codeval)
{
case IECODE_LBUTTON:
 printf("\nMouse Left Button Pressed");
 maybe_mouse_moved();
 break;

case IECODE_RBUTTON:
 printf("\nMouse Right Button Pressed");
 maybe_mouse_moved();
 break;

case (IECODE_LBUTTON + IECODE_UP_PREFIX):
 printf("\nMouse Left Button Released");
 maybe_mouse_moved();
 break;

case (IECODE_RBUTTON + IECODE_UP_PREFIX):
 printf("\nMouse Right Button Released");
 maybe_mouse_moved();
 break;

case IECODE_NOBUTTON:
 timeouts++; /* after 2 minutes, dump program if
 * user loses interest */
 movesize = maybe_mouse_moved();
 if(movesize == 0)
 {
 printf("\n30 seconds passed, no trigger events");
 }
 else if(movesize < XMOVE && movesize < YMOVE)
 {
 printf("\n(Even though less than trigger count,");
 printf("\n reporting mouse move at the selected");
 printf("\n timing interval for user info");
 }
}

```

```

 }
 break;
 default:
 break;
 }
}

set_controller_type(GPCT_NOCONTROLLER);

CloseDevice(game_io_msg);
DeleteStdIO(game_io_msg);
DeletePort(game_msg_port);

printf("\nExiting program... 2 minutes with no activity sensed\n1 > ");
return(0);
}

/* if mouse didn't move far enough to trigger a report, then caller
 * will also report that 30 seconds (1800 vblanks) has elapsed
 */

int maybe_mouse_moved()
{
 int xmove, ymove;
 xmove = game_data->ie_X;
 ymove = game_data->ie_Y;

 if(xmove != 0 || ymove != 0)
 {
 printf("\nMouse Moved by X-value %ld, Y-value %ld",
 xmove, ymove);
 timeouts = 0;
 }
 if(xmove < 0) xmove = -xmove;
 if(ymove < 0) ymove = -ymove;

 return(MAX(xmove,ymove));
}

int set_controller_type(type)
SHORT type;
{
 /* set type of controller to mouse */
 game_io_msg->io_Command = GPD_SETCTYPE;
}

```

```

 *gamedata = type;

 /* set it up */
 /* this command doesn't wait... returns immediately */
 SendIO(game_io_msg);

 WaitPort(game_msg_port);
 GetMsg(game_msg_port);
 return(0);
}

```

## JOYSTICK PROGRAM

```

/* ***** */
* joystick test, for right game port on the Amiga.

* Notes: The right port is used for this test because the input.device task is
* busy continuously with the lefthand port, feeding input events to Intuition or
* console devices. If Intuition is not activated (applications that take over the
* whole machine may decide not to activate Intuition) and no console device is
* activated either, the input device will never activate, allowing the application
* free rein to use either the left OR the right hand joystick/mouse port. If
* either Intuition or the console device is activated, the lefthand port will
* yield, at best, every alternate input event to an external application such as
* this test program. This will undoubtedly mess up either of the two applications
* and should therefore be avoided. It was ok to use the right port in this case,
* because the system has no particular interest in monitoring it.

***** */

#include <exec/types.h>
#include <exec/devices.h>
#include <graphics/gfx.h>
#include <devices/gameport.h>
#include <devices/inputevent.h>

LONG GfxBase=0;

#define XMOVE 10
#define YMOVE 10
#define MAX(m,n) (m > n ? m : n)
#define FOREVER for(;;)
struct InputEvent *game_data; /* pointer into the returned data area
 * where input event has been sent */

```

```

SHORT error;

struct IOStdReq *game_io_msg;

BYTE gamebuffer[sizeof(struct InputEvent)];
BYTE *gamebuff;

SHORT testval;
SHORT codevalue;

struct MsgPort *game_msg_port;

SHORT movesize;
extern struct MsgPort *CreatePort();
extern struct IOStdReq *CreateStdIO();

SHORT codeval, timeouts;

main()
{
 int events_reported;
 events_reported = 0;
 printf(" Joystick Demo\n");
 printf("\nPlug a Joystick Into Right Port\n");
 printf("\nThen move the stick and click its buttons");

 /* point to first location in game buffer */
 gamebuff = &gamebuffer[0];

 /* SYSTEM DEVICE COMMUNICATIONS SUPPORT SETUP ROUTINES ***** */

 /* provide a port for the IO response */
 game_msg_port = CreatePort(0,0);
 if(game_msg_port == 0)
 {
 printf("\nError While Performing CreatePort");
 exit(-1);
 }

 /* make an io request block for communicating with the gameport */
 game_io_msg = CreateStdIO(game_msg_port);

 if(game_io_msg == 0)
 {
 printf("\nError While Performing CreateStdIO");
 }
}

```

```

 DeletePort(game_msg_port);
 exit(-2);
}
/* ***** */
/* OPEN THE DEVICE */

/* open the device for access, unit 1 is right port */
error = OpenDevice("gameport.device",1,game_io_msg,0);

if(error != 0)
{
 printf("\nError while opening the device, exiting");
 DeleteStdIO(game_io_msg);
 DeletePort(game_msg_port);
 exit(-3);
}
/* ***** */
/* SET THE DEVICE TYPE */

game_data = (struct InputEvent *)gamebuffer;

/* test the joystick in this loop */

if (set_controller_type(GPCT_ABSJOYSTICK) != 0)
{
 printf("\nError while trying to set GPCT_ABSJOYSTICK");
 DeleteStdIO(game_io_msg);
 DeletePort(game_msg_port);
 exit(-4);
}
/* ***** */
/* SET THE DEVICE TRIGGER */
if (set_controller_trigger() != 0)
{
 printf("\nError while trying to set controller trigger");
 DeleteStdIO(game_io_msg);
 DeletePort(game_msg_port);
 exit(-4);
}
/* ***** */
/* TELL USER WHAT YOU WILL BE DOING */

printf("\nI will report: \n");
printf("\n Stick X or Y moves");
printf("\n Button presses (along with stick moves if any)");

```



```

/* ***** */
/* SETUP THE IO MESSAGE BLOCK FOR THE ACTUAL DATA READ */

/* from now on, just read input events into the input buffer, one at a
 * time; read-event waits for the preset conditions */

game_io_msg->io_Command = GPD_READEVENT;
game_io_msg->io_Data = (APTR)gamebuffer;

/* read one event each time we go back to the gameport */
game_io_msg->io_Length = sizeof(struct InputEvent);

/* don't use quick io */
game_io_msg->io_Flags = 0;

/* ***** */
/* LOOP FOREVER */

FOREVER
{
 /* read one event each time we go back to the gameport */
 game_io_msg->io_Length = sizeof(struct InputEvent);

 printf("\n Waiting For Joystick Report\n");
 SendIO(game_io_msg);
 WaitPort(game_msg_port);
 /* this is NOT a busy wait... it is a task-sleep */
 GetMsg(game_msg_port);

 codevalue = game_data->ie_Code;

 if(codevalue == IECODE_LBUTTON)
 printf("\nFire Button pressed");
 if(codevalue == (IECODE_LBUTTON + IECODE_UP_PREFIX))
 printf("\nFire Button released");

 which_direction();
 showbugs();
 if (events_reported++>12) break;
}

set_controller_type(GPCT_NOCONTROLLER);

CloseDevice(game_io_msg);
DeleteStdIO(game_io_msg);

```

```

DeletePort(game_msg_port);

printf("\nExiting program... 12 events reported.\n1 > ");
return(0);
}

int which_direction()
{
 SHORT xmove, ymove;
 xmove = game_data->ie_X;
 ymove = game_data->ie_Y;

 switch(ymove)
 {
 case (-1):
 printf("\nForward");
 break;
 case (1):
 printf("\nBack");
 break;
 default:
 break;
 }
 switch(xmove)
 {
 case (-1):
 printf("\nLeft");
 break;
 case (1):
 printf("\nRight");
 break;
 default:
 break;
 }
 return(0);
}

int set_controller_type(type)
SHORT type;
{
 game_io_msg->io_Command = GPD_SETCTYPE;
 /* set type of controller to mouse */
 game_io_msg->io_Length = 1;
 game_io_msg->io_Data = (APTR)gamebuff;
 *gamebuff = type;
}

```

```

SendIO(game_io_msg);
/* set it up */
/* this command doesn't wait... returns immediately */
WaitPort(game_msg_port);
GetMsg(game_msg_port);
return((int)game_io_msg->io_Error);
}

int set_controller_trigger()
{
 struct GamePortTrigger gpt;

 game_io_msg->io_Command = GPD_SETTRIGGER;
 game_io_msg->io_Length = sizeof(gpt);
 game_io_msg->io_Data = (APTR)&gpt;
 gpt.gpt_Keys = GPTF_UPKEYS+GPTF_DOWNKEYS;
 gpt.gpt_Timeout = 0;
 gpt.gpt_XDelta = 1;
 gpt.gpt_YDelta = 1;

 return(DoIO(game_io_msg));
}

showbugs()
{
 struct InputEvent *e;

 e = (struct InputEvent *)&gamebuffer[0];
 /* where the input event gets placed */
 printf("\nie_Class = %lx",e->ie_Class);
 printf("\nie_SubClass = %lx",e->ie_SubClass);
 printf("\nie_Code = %lx", e->ie_Code);
 printf("\nie_Qualifier = %lx",e->ie_Qualifier);
 printf("\nie_X = %ld", e->ie_X);
 printf("\nie_Y = %ld", e->ie_Y);
 printf("\nie_TimeStamp(seconds) = %lx", e->ie_TimeStamp.tv_secs);
 return(0);
}

```

## Chapter 12

### Narrator Device

This chapter provides routines for accessing both the narrator device and the translator library and shows how some of the parameters passed to the device can affect the output. In addition, this chapter contains a nontechnical explanation of how to effectively utilize the speech device. A more technical explanation is also provided for those who may be interested in how the speech is actually produced.

## Introduction

Two different subsystems comprise the speech system on the Amiga. They are the *narrator device*, which communicates with the audio device to actually produce human-like speech, and the *translator library*, which contains a routine that translates English text into phonemes suitable for the narrator device.

## The Translator Library

The translator library provides a single routine, named **Translate()**, that converts an English language string into a phonetic string. To use this function, you must first open the library.

Setting a global variable, **TranslatorBase**, to the value returned from the call to **OpenLibrary()** enables the Amiga linker to correctly locate the translator library:

```
struct Library *TranslatorBase;
...
TranslatorBase = OpenLibrary("translator.library",REVISION);
if(TranslatorBase == NULL) exit (CANT_OPEN_TRANSLATOR);
```

Note that for the **OpenLibrary()** call to succeed, the directory currently assigned by AmigaDOS as *LIBS:* must contain *translator.library*.

### USING THE TRANSLATE FUNCTION

Once the library is open, you can call the translate function:

```
UBYTE *sampleinput; /* pointer to sample input string */
UBYTE outputstring[500]; /* place to put the translation */
SHORT rtnCode; /* return code from function */

sampleinput = "this is a test"; /* a test string of 14 characters */
rtnCode = Translate(sampleinput,14,outputstring,500);
```

The input string will be translated into its phoneme equivalent and can be used to feed the narrator device. If you receive a nonzero return code, you haven't provided enough output buffer space to hold the entire translation. In this case, the **Translate()** function breaks the translation at the end of a word in the input stream and returns the position in the input stream at which the translation ended. You can use the output buffer, then call the **Translate()** function again, starting at this original ending position, to continue the translation where you left off.

Note, however, that the value returned is *negative*. Therefore, you must use **-rtnCode** as the starting point for a new translation.

## ADDITIONAL NOTES ABOUT TRANSLATE

The English language has many words that do not sound the same as they are spelled. The translator library has an exception table that it consults as the translation progresses. Words that are not in the exception table are translated literally. Therefore, it is possible that certain words will not translate well. You can improve the quality of the translation by handling those words on your own, using the tutorial information included at the end of this chapter.

As with all other libraries of routines, if you have opened the translator library for use, be sure to close it before your program exits. If the system needs memory resources, it can then expel closed libraries to gain additional space.

## The Narrator Device

The narrator device on the Amiga provides two basic functions:

- o You can write to the device and ask it to speak a phoneme-encoded string in a specific manner—pitch, male/female, various speaking rates, and so on.
- o You can read from the device. As it speaks, the device can generate mouth shapes for you and you can use the shapes to perform a graphics rendering of a face and mouth.

## OPENING THE NARRATOR DEVICE

To use the narrator device, you must first open the device. The narrator device is disk-resident. For the **OpenDevice()** call to succeed, the narrator device must be present in the directory currently assigned by AmigaDOS to the *DEVS:* directory.

To communicate with the narrator device, like any other device, you must pass an **IORequest** block to **OpenDevice()**. The block used by the narrator device for a write is a special format called a **narrator\_rb**. The block used for a read is also a special format, called a **mouth\_rb**. Both blocks are described in the sections that follow. A sample **OpenDevice()** sequence for the narrator device follows. Notice that two request blocks are created, one for writing to the device and one for reading from it. For brevity, the error checking is left out of this short example. It is, however, utilized in the sample program later on.

```

struct narrator_rb *writeNarrator;
struct narrator_rb *readNarrator;
writeport = CreatePort(0,0);
readport = CreatePort(0,0);
writeNarrator = (struct narrator_rb *)CreateExtIO(writeport,
 sizeof(struct narrator_rb));
readNarrator = (struct narrator_rb *)CreateExtIO(readport,
 sizeof(struct narrator_rb));

```

The routine `CreateExtIO()` is in the "Other Routines" appendix of the *Amiga ROM Kernel Reference Manual: Exec*. `CreatePort()` is contained in *amiga.lib* and can be accessed by linking your program to *amiga.lib*.

## CONTENTS OF THE WRITE REQUEST BLOCK

You can control several characteristics of the speech, as indicated in the narrator request block structure shown below.

```

struct narrator_rb {
 struct IOStdReq message; /* Standard IORB */
 UWORD rate; /* Speaking rate (words/minute) */
 UWORD pitch; /* Baseline pitch in Hertz */
 UWORD mode; /* Pitch mode */
 UWORD sex; /* Sex of voice */
 UBYTE *ch_masks; /* Pointer to audio alloc maps */
 UWORD nm_masks; /* Number of audio alloc maps */
 UWORD volume; /* Volume. 0 (off) thru 64 */
 UWORD sampfreq; /* Audio sampling freq */
 UBYTE mouths; /* If non-zero, generate mouths */
 UBYTE chanmask; /* Which ch mask used (internal) */
 UBYTE numchan; /* Num ch masks used (internal) */
 UBYTE pad; /* For alignment */
};

```

where

### rate

is the speed in words per minute that you wish it to speak.

### pitch

is the baseline pitch. If you are using an expressive voice rather than a monotone, the pitch will vary above and below this baseline pitch.

**mode**

determines whether you have a monotone or expressive voice.

**sex**

determines if the voice is male or female.

**ch\_masks, nm\_masks, volume, sampfreq**

are described in the chapter called "Audio Device."

**mouths**

is set to nonzero before starting a write if you want to read mouths using the read command while the system is speaking.

**chanmask, numchan, pad**

are for system use only.

The system default values are shown in the files *devices/narrator.h* and *devices/narrator.i*. When you call **OpenDevice()**, the system initializes the request block to the default values. If you want other than the defaults, you must change them *after* the device is open.

## CONTENTS OF THE READ REQUEST

The **mouth\_rb** data structure follows. Notice that it is an extended form of the **narrator\_rb** structure.

```

struct mouth_rb {
 struct narrator_rb voice; /* Speech IORB */
 UBYTE width; /* Width (returned value) */
 UBYTE height; /* Height (returned value) */
 UBYTE shape; /* Internal use, do not modify */
 UBYTE pad; /* For alignment */
};

```

The fields **width** and **height** will, on completion of a read-request, contain an integer value proportional to the mouth width and height that are appropriate to the phoneme currently being spoken. When you send a read request, the system does not return a response until one of two things happens. Either a different mouth size is available (this prevents you from drawing and redrawing the same shape or having to check whether or not it is the same) or the speaking has completed. You must check the error return field when the read request block is returned to determine if the request block contains a new mouth shape or simply is returning status of **ND\_NoWrite** (no write in progress, all speech ended for this request).



## OPENING THE NARRATOR DEVICE

This section demonstrates opening the device as well as synchronizing a read request so that it responds only to the write request for which the device is opened. You can read the mouth shapes only if the write request contains the same unit number and a write is currently in progress; the system returns an error if the numbers don't match or if the write has completed. Note again that error checking is deferred to the example program at the end of the chapter.

```
SHORT openError;
```

```
openError = OpenDevice("narrator.device",0,writeNarrator,0);
 /* after error checking, synchronize the read and write requests */
readNarrator->narrator_rb.message.io_Device =
 writeNarrator->message.io_Device; /* copy device info */
readNarrator->narrator_rb.message.io_Unit =
 writeNarrator->message.io_Unit; /* copy unit info */
```

At this point, it is acceptable to change the default values before issuing a write.

More details about what **OpenDevice()** performs are contained in the narrator device summary pages.

## PERFORMING A WRITE AND A READ

You normally perform a write command by using the functions **BeginIO()** or **SendIO()** to transmit the request block to the narrator device. This allows the narrator's task to begin the I/O, while your task is free to do something else. The something else may be issuing a series of read commands to the device to determine mouth shapes and drawing them on-screen. The following sample set of function calls implements both the write and read commands in a single loop. Again, error checking is deferred to the sample program.

```

SHORT readError;

writeNarrator->message.io_Length = strlen(outputstring);
 /* tell it how many characters the translate function returned */
writeNarrator->message.io_Data = outputstring;
 /* tell it where to find the string to speak */
SendIO(writeNarrator);
 /* return immediately, run tasks concurrently */

readNarrator->voice.message.io_Error = 0;
while((readError = readNarrator->voice.message.io_Error) !=
 ND_NoWrite)
{
 DoIO(readNarrator);
 /* put task to sleep waiting for a different mouth shape or
 * return of the message block with the error field showing
 * no write in progress
 */
 DrawMouth(readNarrator->width,readNarrator->height);
 /* user's own unique routine, not provided here */
}
GetMsg(writeport); /* remove the write message from the
 * writeport so that it can be reused */

```

The loop continues to send read requests to the narrator device until the speech output has ended. `DoIO()` automatically removes the read request block from the readport for reuse. `SendIO()` is used to transmit the write request. When it completes, the write request will be appended to the writeport, and must be removed before it can be reused.

## Sample Program

The following sample program uses the system default values returned from the `OpenDevice()` call. It translates and speaks a single phrase.

```

#include "exec/types.h"
#include "exec/exec.h"

#include "exec/nodes.h"
#include "exec/lists.h"
#include "exec/memory.h"
#include "exec/interrupts.h"
#include "exec/ports.h"
#include "exec/libraries.h"
#include "exec/io.h"
#include "exec/tasks.h"
#include "exec/execbase.h"

#include "devices/narrator.h"
#include "libraries/translator.h"

struct MsgPort *readport=0;
struct MsgPort *writeport=0;

extern struct MsgPort *CreatePort();
extern struct IORequest *CreateExtIO();

struct narrator_rb *writeNarrator=0;
struct mouth_rb *readNarrator=0;
struct Library *TranslatorBase=0;
UBYTE *sampleinput; /* pointer to sample input string */
UBYTE outputstring[500]; /* place to put the translation */
SHORT rtnCode; /* return code from function */
SHORT readError;
SHORT writeError;
SHORT error;
BYTE audChanMasks[4] = { 3,5,10,12 }; /* which channels to use */

#define CANT_OPEN_TRANSLATOR -100
#define CANT_OPEN_NARRATOR -200
#define CREATE_PORT_PROBLEMS -300
#define CREATE_IO_PROBLEMS -400
#define CANT_PERFORM_WRITE -500
#define REVISION 1

extern struct Library *OpenLibrary();

main()
{
 TranslatorBase = OpenLibrary("translator.library",REVISION);
}

```

```

if(TranslatorBase == NULL) exit (CANT_OPEN_TRANSLATOR);
sampleinput = "this is a test"; /* a test string of 14 characters */
rtnCode = Translate(sampleinput,14,outputstring,500);
error = rtnCode + 100;
if(rtnCode != 0) goto cleanup0;

writeport = CreatePort(0,0);
if(writeport == NULL) { error=CREATE_PORT_PROBLEMS; goto cleanup1; }
readport = CreatePort(0,0);
if(readport == NULL) { error=CREATE_PORT_PROBLEMS; goto cleanup2; }
writeNarrator = (struct narrator_rb *)CreateExtIO(writeport,
 sizeof(struct narrator_rb));
if(writeNarrator == NULL) { error=CREATE_IO_PROBLEMS; goto cleanup3; }
readNarrator = (struct mouth_rb *)CreateExtIO(readport,
 sizeof(struct mouth_rb));

if(readNarrator == NULL) { error=CREATE_IO_PROBLEMS; goto cleanup4; }
/* SET UP PARAMETERS FOR WRITE-MESSAGE TO THE NARRATOR DEVICE */

/* show where to find the channel masks */
writeNarrator->ch_masks = (audChanMasks);

/* and tell it how many of them there are */
writeNarrator->nm_masks = sizeof(audChanMasks);

/* tell it where to find the string to speak */
writeNarrator->message.io_Data = (APTR)outputstring;

/* tell it how many characters the translate function returned */
writeNarrator->message.io_Length = strlen(outputstring);

/* if nonzero, asks that mouths be calculated during speech */
writeNarrator->mouths = 1;

/* tell it this is a write-command */
writeNarrator->message.io_Command = CMD_WRITE;

/* Open the device */

error = OpenDevice("narrator.device", 0, writeNarrator, 0);
if(error != 0) goto cleanup4;

/* SET UP PARAMETERS FOR READ-MESSAGE TO THE NARRATOR DEVICE */

/* tell narrator for whose speech a mouth is to be generated */

```

```

readNarrator->voice.message.io_Device =
 writeNarrator->message.io_Device;
readNarrator->voice.message.io_Unit =
 writeNarrator->message.io_Unit;

readNarrator->width = 0;
readNarrator->height = 0; /* initial mouth parameters */

readNarrator->voice.message.io_Command = CMD_READ;
 /* initial error value */
readNarrator->voice.message.io_Error = 0;

/* Send an asynchronous write request to the device */

writeError = SendIO(writeNarrator);
if(writeError != NULL) { error=CANT_PERFORM_WRITE; goto cleanup5; }
/* return immediately, run tasks concurrently */

/* keep sending reads until it comes back saying "no write in progress" */

while((readError = readNarrator->voice.message.io_Error) !=
 ND_NoWrite)
{
 DoIO(readNarrator);
 /* put task to sleep waiting for a different mouth shape
 * or return of the message block with the error field
 * showing no write in progress
 */
 DrawMouth(readNarrator->width,readNarrator->height);
 /* user's own unique routine, not provided here */
}

Delay(30);

rtnCode = Translate("No it is not",13,outputstring,500);
writeNarrator->sex = FEMALE;
writeNarrator->pitch = MAXPITCH; /* raise pitch from default value */
writeNarrator->message.io_Data = (APTR)outputstring;
writeNarrator->message.io_Length = strlen(outputstring);
DoIO(writeNarrator);

Delay(30);

rtnCode = Translate("Please! I am speaking now!",26,outputstring,500);
writeNarrator->sex = MALE;

```

```

writeNarrator->pitch = DEFPITCH;
writeNarrator->message.io_Data = (APTR)outputstring;
writeNarrator->message.io_Length = strlen(outputstring);
DoIO(writeNarrator);

Delay(30);

rtnCode = Translate(
 "Well, you are not very interesting, so I am going home!",
 55,outputstring,500);
writeNarrator->sex = FEMALE;
writeNarrator->pitch = MAXPITCH;
writeNarrator->message.io_Data = (APTR)outputstring;
writeNarrator->message.io_Length = strlen(outputstring);
DoIO(writeNarrator);

Delay(30);

rtnCode = Translate("Bye Bye",7,outputstring,500);
writeNarrator->sex = MALE;
writeNarrator->pitch = DEFPITCH;
writeNarrator->rate = 7; /* slow him down */
writeNarrator->message.io_Data = (APTR)outputstring;
writeNarrator->message.io_Length = strlen(outputstring);
DoIO(writeNarrator);

cleanup5:
if(writeNarrator != 0)
 CloseDevice(writeNarrator);
 /* terminate access to the device */

/* now return system memory to the memory allocator */

cleanup4:
if(readNarrator != 0)
 DeleteExtIO(readNarrator,sizeof(struct mouth_rb));
cleanup3:
if(writeNarrator != 0)
 DeleteExtIO(writeNarrator,sizeof(struct narrator_rb));
cleanup2:
if(readport != 0)
 DeletePort(readport);
cleanup1:
if(writeport != 0)
 DeletePort(writeport);

```

```

cleanup0:
 if(TranslatorBase != 0)
 CloseLibrary(TranslatorBase);
 /* terminate access to the library */

 if(error != 0) exit(error);
} /* end of test */

DrawMouth(w,h)
SHORT w,h;
{ return(0); /* dummy routine */ }

int strlen(string)
char *string;
{
 int i,length;
 length = -1;
 for(i=0; i<256; i++) /* 256 characters max length at this time */
 {
 if(*string++ == ' ') { length = i+1; break; };
 }
 return(length);
}

```

The loop continues to send read requests to the narrator device until the write request has completed. Then the program cleans up and exits.

You can experiment with the narrator device by using values other than the default, changing them before the write command is sent to the device.

## How to Write Phonetically for Narrator

This section describes in detail the procedure used to specify phonetic strings to the *Narrator* speech synthesizer. No previous experience with phonetics is required. The only thing you may need is a good pronouncing dictionary for those times when you doubt your own ears. You do not have to learn a foreign language or computer language. You are just going to learn how to write down the English that comes out of your own mouth. In writing phonetically you do not have to know how a word is spelled, just how it is said.

Narrator works on utterances at the sentence level. Even if you want to say only one word, Narrator will treat it as a complete sentence. Therefore, Narrator wants one of two punctuation marks to appear at the end of every sentence—a period (.) or a question mark (?). If no

punctuation appears at the end of a string, Narrator will append a period to it. The period is used for almost all utterances and will cause a final fall in pitch to occur at the end of a sentence. The question mark is used at the end of yes/no questions only, and results in a final rise in pitch. For example, the question, *Do you enjoy using your Amiga?* would take a question mark at the end because the answer to the question is either yes or no. The question, *What is your favorite color?* would not take a question mark and should be followed by a period. Narrator recognizes other punctuation marks as well, but these are left for later discussion.

## PHONETIC SPELLING

Utterances are usually written phonetically using an alphabet of symbols known as I.P.A. (for "International Phonetic Alphabet"). This alphabet is found at the front of most good dictionaries. The symbols can be hard to learn and are not available on computer keyboards, so the Advanced Research Projects Agency (ARPA) came up with *Arpabet*, a way of representing each symbol using one or two upper-case letters. Narrator uses an expanded version of Arpabet to specify phonetic sounds.

A phonetic sound, or *phoneme*, is a basic speech sound, almost a speech atom. Working backwards, sentences can be broken into words, words into syllables, and syllables into phonemes. The word *cat* has three letters and (coincidentally) three phonemes. Looking at the table of phonemes we find the three sounds that make up the word *cat*. They are K, AE, and T, written as KAET. The word *cent* translates as S, EH, N and T, or SEHNT. Notice that both words begin with a *c* but because the *c* says *k* in *cat* we use the phoneme K. In *cent* the *c* says *s* so we use the phoneme S. You may also have noticed that there is no C phoneme.

The above example illustrates that a word rarely sounds like it looks in English spelling. These examples introduce you to a very important concept: spell it like it sounds, not like it looks.

## CHOOSING THE RIGHT VOWEL

Phonemes, like letters, are divided into the two categories of vowels and consonants. Loosely defined, a vowel is a continuous sound made with the vocal cords vibrating and air exiting the mouth (as opposed to the nose). All vowels use a two-letter code. A consonant is any other sound, such as those made by rushing air (like S or TH), or by interruptions in air flow by the lips or tongue (like B or T). Consonants use a one- or two-letter code.

In English we write with only five vowels: a, e, i, o and u. It would be easy if we only *said* five vowels. Unfortunately, we say more than 15 vowels. Narrator provides for most of them. You choose the proper vowel by listening. Say the word out loud, perhaps extending the vowel sound you want to hear. Compare the sound you are making to the sounds made by the vowels in the example words to the right of the phoneme list. For example, the *a* in *apple* sounds the same as the *a* in *cat*, not like the *as* in *Amiga*, *talk*, or *made*. Notice also that some of the



example words in the list do not even use any of the same letters contained in the phoneme code; for example, AA as in *hot*.

Vowels are divided into two groups: those that maintain the same sound throughout their durations and those that change their sound. The ones that change are called *diphthongs*. Some of us were taught the terms *long* and *short* to describe vowel sounds. Diphthongs fall into the long category, but these two terms are inadequate to fully differentiate between vowels and should be avoided. The diphthongs are the last six vowels listed in the table. Say the word *made* out loud very slowly. Notice how the *a* starts out like the *e* in *bet* but ends up like the *e* in *beet*. The *a* therefore is a diphthong in this word and we would use EY to represent it. Some speech synthesis systems require you to specify the changing sounds in diphthongs as separate elements, but Narrator takes care of the assembly of diphthongal sounds for you.

## CHOOSING THE RIGHT CONSONANT

Consonants are divided into many categories by phoneticians, but we need not concern ourselves with most of them. Picking the correct consonant is very easy if you pay attention to just two categories: voiced and unvoiced. A voiced consonant is made with the vocal cords vibrating, and an unvoiced one is made when the vocal cords are silent. Sometimes English uses the same letter combinations to represent both. Compare the *th* in *thin* and in *then*. Notice that the first is made with air rushing between the tongue and upper teeth. In the second, the vocal cords are vibrating also. The voiced *th* phoneme is DH, the unvoiced is TH. Therefore, *thin* is spelled TH, IH, N or THIHN, and *then* is spelled DH, EH, N or DHEHN. A sound that is particularly subject to mistakes is voiced and unvoiced *s* spelled Z or S. To put it clearly, *bats* ends in S, *suds* ends in Z. What kind of *s* does *closet* have? How about *close*? Say all of these words out loud to find out. Actually *close* changes its meaning when the *s* is voiced or unvoiced: *I love to be close to you.* versus *What time do you close?*

Another sound that causes some confusion is the *r* sound. There are two different r-like phonemes in the Narrator alphabet: R under the consonants and ER under the vowels. Which one do you use? Use ER if the *r* sound is the vowel sound in the syllable. Words that take ER are *absurd*, *computer* and *flirt*. Use R if the *r* sound precedes or follows another vowel sound in that syllable, such as in *car*, *write*, or *craft*. *Rooster* uses both kinds of *r*. Can you tell which is which?

## CONTRACTIONS AND SPECIAL SYMBOLS

There are several phoneme combinations that appear very often in English words. Some of these are caused by our laziness in pronunciation. Take the word *connector* for example. The *o* in the first syllable is almost swallowed out of existence. You would not use the AA phoneme; you would use the AX instead. It is because of this *relaxation* of vowels that we find ourselves using AX and IX very often. Since this relaxation frequently occurs before l, m and n, Narrator

has a shortcut for typing these combinations. Instead of *personal* being spelled PERSIXNAXL, we can spell it PERSINUL, making it a little more readable. *Anomaly* goes from AXNAAMAXLIY to UNAAMULIY, and KAAMBIXNEYSHIXN becomes KAAMBINEYSHIN for *combination*. It may be hard to decide whether to use the AX or IX brand of relaxed vowel. The only way to find out is to try both and see which sounds best.

Other special symbols are used internally by Narrator. Sometimes they are inserted into or substituted for part of your input sentence. You can type them in directly if you wish. The most useful is probably the Q or glottal stop; an interruption of air flow in the glottis. The word *Atlantic* has one between the *t* and the *l*. Narrator knows there should be a glottal stop there and saves you the trouble of typing it. But Narrator is only close to perfect, so sometimes a word or word pair might slip by that would have sounded better with a Q stuck in someplace.

## STRESS AND INTONATION

It is not enough to tell Narrator what you want said. For the best results you must also tell Narrator how you want it said. In this way you can alter a sentence's meaning, stress important words, and specify the proper accents in polysyllabic words. These things improve the naturalness and thus the intelligibility of Narrator's spoken output.

Stress and intonation are specified by the single digits 1-9 following a vowel phoneme code. Stress and intonation are two different things but are specified by a single number. Stress is, among other things, the elongation of a syllable. Because a syllable is either stressed or not, the presence of a number after the vowel in a syllable indicates stress on that syllable. The value of the number indicates the intonation. From this point onward, these numbers will be referred to as *stress marks*. Intonation here means the pitch pattern or contour of an utterance. The higher the stress mark, the higher the potential for an accent in pitch (a rise and fall). A sentence's basic contour is comprised of a quickly rising pitch gesture up to the first stressed syllable in the sentence, followed by a slowly declining tone throughout the sentence, and finally a quick fall to a low pitch on the last syllable. The presence of additional stressed syllables causes the pitch to break its slow, declining pattern with rises and falls around each stressed syllable. Narrator uses a very sophisticated procedure to generate natural pitch contours based on how you mark the stressed syllables.

## HOW AND WHERE TO PUT THE STRESS MARKS

The stress marks go immediately to the right of vowel phoneme codes. The word *cat* has its stress marked after the AE so we get KAE5T or KAE9T. You generally have no choice about the location of a number; there is definitely a right and wrong location. Either a number should go after a vowel or it should not. Narrator will not flag an error if you forget to put a stress mark in or if you place one on the wrong vowel. It will only tell you if a stress mark is in the wrong place, such as after a consonant.

The rules for placing stress marks are as follows:

- o Always place a stress mark in a *content* word. A content word is one that contains some meaning. Nouns, verbs, and adjectives are all content words. *Boat, huge, tonsils* and *hypertensive* are all content words; they tell the listener what you are talking about. Words like *but, the, if* and *is* are not content words. They do not convey any real-world meaning at all but are required to make the sentence function. Thus, they are given the name *function words*.
- o Always place a stress mark on the accented syllable(s) of polysyllabic words, whether they are content or function words. A polysyllabic word is any word of more than one syllable. *Commodore* has its stress (or accent as it is often called) on the first syllable and would be spelled KAA5MAXDOHR. *Computer* is stressed on the second syllable, producing KUMPYUW5TER.

If you are in doubt about which syllable gets the stress, look the word up in a dictionary and you will find an accent mark over the stressed syllable. If more than one syllable in a word receives stress, they usually are not of equal value. These are referred to as primary and secondary stresses. The word *understand* has its first and last syllables stressed, with *stand* getting primary stress and *un* secondary, which produces AH1NDERSTAE4ND. Syllables with secondary stress should be marked with a value of only 1 or 2.

Compound words (words with more than one root) such as *base/ball, soft/ware, lunch/wagon*, and *house/boat* can be written as one word but should be thought of as separate words when marking stress. Thus, *lunchwagon* would be spelled LAH5NCHWAE2GIN. Notice that *lunch* got a higher stress mark than *wagon*. This is common in compound words; the first word usually receives the primary stress.

## WHAT STRESS VALUE DO I USE?

If you get the spelling and stress mark positions correct, you are 95 percent of the way to a good sounding sentence. The next thing to do is decide on the stress mark values. They can be roughly related to parts of speech, and you can use table 12-1 as a guide to assigning values.

Table 12-1: Recommended Stress Values

| Part of Speech   | Stress Value |               |
|------------------|--------------|---------------|
| Nouns            | 5            |               |
| Pronouns         | 3            |               |
| Verbs            | 4            |               |
| Adjectives       | 5            |               |
| Adverbs          | 7            |               |
| Quantifiers      | 7            |               |
| Exclamations     | 9            |               |
| Articles         | 0            | (no stress)   |
| Prepositions     | 0            |               |
| Conjunctions     | 0            |               |
| Secondary stress | 1            | (sometimes 2) |

The above values merely suggest a range. If you want attention directed to a certain word, raise its value. If you want to downplay a word, lower it. Sometimes even a function word can be the focus of a sentence. It is quite conceivable that the word “to” in the sentence “Please deliver this to Mr. Smith.” could receive a stress mark of 9. This would add focus to the word “to” indicating that the item should be delivered to Mr. Smith in person.

## PUNCTUATION

In addition to the period or question mark that is required at the end of a sentence, Narrator recognizes several other punctuation marks: dashes, commas, and parentheses. The comma goes where you would normally put a comma in an English sentence. It causes Narrator to pause with a slightly rising pitch, indicating that there is more to come. The use of additional commas—that is, more than would be required for written English—is often helpful. They serve to set clauses off from one another. There is a tendency for a listener to lose track of the meaning of a sentence if the words run together. Read your sentence aloud while pretending to be a newscaster. The locations for additional commas should leap out at you.

The dash serves almost the same purpose as the comma, except that the dash does not cause the pitch to rise so severely. A rule of thumb is: Use dashes to divide phrases, commas to divide clauses. For a definition of these terms, consult a high school English book.

Parentheses provide additional information to Narrator’s intonation routine. They should be put around noun phrases of two or more content words. This means that the noun phrase, “a giant yacht” should be surrounded with parentheses because it contains two content words, *giant* and *yacht*. The phrase *my friend* should not have parentheses around it because it contains only one content word. Noun phrases can get pretty big, like “the silliest guy I ever saw”

or "a big basket of fruit and nuts." The parentheses really are most effective around these large phrases; the smaller ones can sometimes go without. The effect of parentheses is subtle, and in some sentences you might not even notice their presence. In sentences of great length, however, they help provide for a very natural contour.

## HINTS FOR INTELLIGIBILITY

There are a few tricks you can use to improve the intelligibility of a sentence. Often, a polysyllabic word is more recognizable than a monosyllabic word. For instance, instead of saying *huge*, say *enormous*. The longer version contains information in every syllable, thus giving the listener three times the chance to hear it correctly. This can be taken to extremes, so try not to say things like "This program has a plethora of insects in it."

Another good practice is to keep sentences to an optimal length. Writing for reading and writing for speaking are two different things. Try not to write a sentence that cannot be easily spoken in one breath. Such a sentence tends to give the impression that the speaker has an infinite lung capacity. Try to keep sentences confined to one main idea. A run-on sentence tends to lose its meaning after a while.

New terms should be highly stressed the first time they are heard. If you are doing a tutorial or something similar, stress a new term at its first occurrence. All subsequent occurrences of that term need not be stressed as highly because it is now "old news."

The above techniques are but a few ways to enhance the performance of Narrator. You will probably find some of your own. Have fun.

## EXAMPLE OF ENGLISH AND PHONETIC TEXTS

Cardiomyopathy. I had never heard of it before, but there it was listed as the form of heart disease that felled not one or two but all three of the artificial heart recipients. A little research produced some interesting results. According to an article in the Nov. 8, 1984, *New England Journal of Medicine*, cigarette smoking causes this lethal disease that weakens the heart's pumping power. While the exact mechanism is not clear, Dr. Arthur J. Hartz speculated that nicotine or carbon monoxide in the smoke somehow poisons the heart and leads to heart failure.

KAA1RDIYOWMAYAA5PAXTHIY. AY /HAED NEH1VER HER4D AXV IHT BIXFOH5R,  
BAHT DHEH5R IHT WAHZ - LIH4STIXD AEZ (DHAX FOH5RM AXV /HAA5RT DIHZIY5Z)  
DHAET FEH4LD (NAAT WAH5N OHR TUW5) - BAHT (AO7L THRIY5 AXV DHAX  
AA5RTAXFIHSHUL /HAA5RT RIXSIH5PIYINTS). (AH LIH5TUL RIXSER5CH)  
PROHDUW5ST (SAHM IH5NTRIHSTIHNX RIXZAH5LTS). AHKOH5RDIH5NX TUW (AEN  
AA5RTIHKUL IHN DHAX NOWVEH5MBER EY2TH NAY5NTIYNEYTIYFOH1R NUW  
IY5NXGLIND JER5NUL AXV MEH5DIXSIN), (SIH5GEREHT SMOW5KIHNX) KAO4ZIHZ

(DHIHS LIY5THUL DIHZIY5Z) DHAET WIY4KINZ (DHAX /HAA5RTS PAH4MPIHNX PAW2ER). WAYL (DHIY IHGZAE5KT MEH5KINIXZUM) IHZ NAAT KLIY5R, DAA5KTER AA5RATHER JEY2 /HAA5RTS SPEH5KYULEYTIHD DHAET NIH5KAXTIYN OHR KAA5RBIN MUNAA5KSAYD IHN DHAX SMOW5K - SAH5M/HAW1 POY4ZINZ DHAX /HAA5RT - AEND LIY4DZ TUW (/HAA5RT FEY5LYER).

## CONCLUDING REMARKS

This guide should get you off to a good start in phonetic writing for Narrator. The only way to get really proficient is to practice. Many people become good at it in as little as one day. Others make continual mistakes because they find it hard to let go of the rules of English spelling, so trust your ears.

## The More Technical Explanation

The *SoftVoice* speech synthesis system is a computer model of the human speech production process. It attempts to produce accurately spoken utterances of any English sentence, given only a phonetic representation as input. Another program in the system, *Translator*, derives the required phonetic spelling from English text. Timing and pitch contour are produced automatically by the synthesizer software.

In humans, the physical act of producing speech sounds begins in the lungs. To create a voiced sound, the lungs force air through the vocal folds (sometimes called the vocal cords), which are held under tension and which periodically interrupt the flow of air, thus creating a buzz-like sound. This buzz, which has a spectrum rich in harmonics, then passes through the vocal tract and out the lips, which alters its spectrum drastically. This is because the vocal tract acts as a frequency filter, selectively reinforcing some harmonics and suppressing others.

It is this filtering that gives a speech sound its identity. The amplitude versus frequency graph of the filtering action is called the *vocal tract transfer function*. Changing the shape of the throat, tongue, and mouth retunes the filter system to accent different frequencies.

The sound travels as a pressure wave through the air, and it causes the listener's eardrum to vibrate. The ear and brain of the listener decodes the incoming frequency pattern. From this the listener can subconsciously make a judgment about what physical actions were performed by the speaker to make the sound. Thus the speech chain is completed, the speaker having encoded his physical actions on a buzz via selective filtering and the listener having turned the sound into guesses about physical actions by frequency decoding.

Now that we know how we do it, how does a machine do it? It turns out that the vocal tract is not random, but tends to accentuate energy in narrow regions called *formants*. The formant positions move smoothly as we speak, and it is the formant frequencies to which our ears are sensitive. So, luckily, we do not have to model throat, tongue, teeth and lips with our computer, we can imitate formant action.

A good representation of speech requires up to five formants, but only the lowest three are required for intelligibility. We begin with an oscillator that produces a waveform similar to that which is produced by the vocal folds, and we pass it through a series of resonators, each tuned to a different formant frequency. By controlling the volume and pitch of the oscillator and the frequencies of the resonators, we can produce highly intelligible and natural-sounding speech. Of course the better the model, the better the speech; but more importantly, experience has shown that the better the control of the model's parameters, the better the speech.

Oscillators, volume controls and resonators can all be simulated mathematically in software, and it is by this method that the SoftVoice system operates. The input phonetic string is converted into a series of target values for the various parameters illustrated. A system of rules then operates on the string to determine things such as the duration of each phoneme and the pitch contour. Transitions between target values are created and smoothed to produce natural continuous changes from one sound to the next.

New values are computed for each parameter for every 8 milliseconds of speech, which produces about 120 acoustic changes per second. These values drive a mathematical model of the speech synthesizer. The accuracy of this simulation is quite good. Human speech has more formants than the SoftVoice model, but they are low in energy content.

The human speech production mechanism is a complex and wonderful thing. The more we learn about it, the better we can make our computer simulations. Meanwhile, we can use synthetic speech as yet another computer output device to enhance the man/machine dialogue.

## Table of Phonemes

Table 12-2 lists all the available phonemes.

Table 12-2: Phonemes

Vowels

| Phoneme | Example | Phoneme | Example |
|---------|---------|---------|---------|
| IY      | beet    | IH      | bit     |
| EH      | bet     | AE      | bat     |
| AA      | hot     | AH      | under   |
| AO      | talk    | UH      | look    |
| ER      | bird    | OH      | border  |
| AX*     | about   | IX*     | solid   |

\*AX and IX should never be used in stressed syllables.

Diphthongs

| Phoneme | Example | Phoneme | Example |
|---------|---------|---------|---------|
| EY      | made    | AY      | hide    |
| OY      | boil    | AW      | power   |
| OW      | low     | UW      | crew    |

Consonants

| Phoneme | Example   | Phoneme | Example  |
|---------|-----------|---------|----------|
| R       | red       | L       | yellow   |
| W       | away      | Y       | yellow   |
| M       | men       | N       | men      |
| NX      | sing      | SH      | rush     |
| S       | sail      | TH      | thin     |
| F       | fed       | ZH      | pleasure |
| Z       | has       | DH      | then     |
| V       | very      | J       | judge    |
| CH      | check     | /C      | loch     |
| /H      | hole      | P       | put      |
| B       | but       | T       | toy      |
| D       | dog       | G       | guest    |
| K       | Commodore |         |          |



### Special Symbols

| Phoneme | Example |                |
|---------|---------|----------------|
| DX      | pity    | (tongue flap)  |
| Q       | kitt_en | (glottal stop) |
| QX      | pause   | (silent vowel) |
| RX      | car     | (postvocalic   |
| LX      | call    | R and L)       |

### Contractions

(see text)

|    |   |     |
|----|---|-----|
| UL | = | AXL |
| IL | = | IXL |
| UM | = | AXM |
| IM | = | IXM |
| UN | = | AXN |
| IN | = | IXN |

### Digits and Punctuation

|            |                                                          |
|------------|----------------------------------------------------------|
| Digits 1-9 | Syllabic stress, ranging from secondary through emphatic |
| .          | Period—sentence final character                          |
| ?          | Question mark—sentence final character                   |
| -          | Dash—phrase delimiter                                    |
| ,          | Comma—clause delimiter                                   |
| ( )        | Parentheses—noun phrase delimiters (see text)            |

# Chapter 13

## Serial Device

This chapter describes software access to the serial port. The serial device is accessed via the standard system device-access routines and provides some additional functions specifically appropriate to use of this device.

### Introduction

The serial device can be opened in either exclusive access mode or shared mode. It can be set to transmit and receive many different baud rates (send and receive baud rates are identical). It can support a seven-wire handshaking as well as a three-wire interconnect to a serial hardware

device. Handshaking and access mode must be specified before the serial device is opened. Other serial parameters can be specified using the `SDCMD_SETPARAMS` command after the device has been opened.

## Opening the Serial Device

Typically, you open the serial device by using the following function calls:

```
LONG error;
struct Port *mySerPort;
struct IOExtSer *mySerReq;

/* create a reply port to which serial device can return the request */
mySerPort = CreatePort("mySerial",0);
if(mySerPort == NULL) exit(100); /* can't create port? */

/* create a request block appropriate to serial */
mySerReq = (struct IOExtSer *)CreateExtIO(mySerPort,
 sizeof(struct IOExtSer));
if(mySerReq == NULL) goto cleanup1; /* error during CreateExtIO? */

mySerReq->io_SerFlags = 0;
/* Accept the default, i.e., exclusive Access and XON/XOFF protocol
 * is enabled. Remaining flags all zero, see devices/serial.h
 * for bit-positions. Definitions included in this chapter. */

error = OpenDevice("serial.device",0,mySerReq,0);
if(error != 0) goto cleanup2; /* device not available? */

...
cleanup2:
 DeleteExtIO(mySerReq,sizeof(struct IOExtSer));
cleanup1:
 DeletePort(mySerPort);
```

The routines `CreatePort()` and `DeletePort()` are part of `amiga.lib`. Information about the routines `CreateExtIO()` and `DeleteExtIO()` can be found in the appendixes of the *Amiga ROM Kernel Reference Manual: Exec*.

During the open, the only flags that the serial device pays any attention to are the shared/exclusive-access flag and the seven-wire flag (the seven-wire flag enables RS-232-C DTR/DSR,RTS/CTS handshaking protocol). All other bits in `io_SerFlags` are ignored. However, for consistency, the other flag bits should be set to zero when the device is opened.

When the serial device is opened, it opens the timer device and then allocates an input buffer of the size last used (default and minimum = 512 bytes). As with any of the other serial port parameters, you can later change the value used for the read buffer size with the **SDCMD\_SETPARMS** command. The **OpenDevice()** routine will fill the latest parameter settings into the **io\_Request** block.

Once the serial device is opened, all characters received will be saved, even if there is no current request for them. Note that a parameter change cannot be performed while an I/O request is actually being processed, because it would invalidate request-handling already in progress. Therefore you must use **SDCMD\_SETPARAMS** only when you have no serial I/O requests pending.

## Reading from the Serial Device

You read from the serial device by sending your **IORequest (IOExtSer)** to the device with a read command. You specify how many bytes are to be transferred and where the data is to be placed. Depending on how you have set your parameters, the request may read the requested number of characters or it may terminate early.

Here is a sample read command:

```
char myDataArea[100];
mySerReq->IOSer.io_Data = &myDataArea[0]; /* where to put the data */
mySerReq->IOSer.io_Length = 100; /* read 100 characters */
mySerReq->IOSer.io_Command = CMD_READ; /* say it is a read */
DoIO(mySerReq); /* synchronous request */
```

If you use this example, your task will be put to sleep waiting until the serial device reads 100 bytes (or terminates early) and copies them into your read-buffer. Early termination can be caused by error conditions or by the serial device sensing an end of file condition.

Note that the **io\_Length** value, if set to -1, tells the serial device that you want to read a null-terminated string. The device will read all incoming characters up to and including a byte value of 0x00 in the input stream and will then report to you an **io\_Actual** value that is the actual length of the string, excluding the 0 value. Be aware that you must encounter a 0 value in the input stream before the system fills up the buffer you have specified. The **io\_Length** is, for all practical purposes, indefinite. Therefore, you could potentially overwrite system memory if you never encountered the null termination (zero value byte) in the input stream.

## FIRST ALTERNATIVE MODE FOR READING

As an alternative to **DoIO()** you can use **SendIO()** to transmit the command to the device. In this case, your task can go on to do other things while the serial device is collecting the bytes for you. You can occasionally do a **CheckIO(mySerReq)** to see if the I/O is completed.

```
struct Message *myIO;

/* same code as in above example, except: */
SendIO(mySerReq);

 /* do something */
 /* (user code) */
myIO = CheckIO(mySerReq);
if(myIO != FALSE) goto ioDone; /* this IO is done */

 /* do something else */
 /* (user code) */
WaitIO(mySerReq);
myIO = mySerReq; /* if had to wait, need a value for myIO */
}
ioDone:
 Remove(mySerPort->mp_MsgList,myIO);
/* use the Remove function rather than the GetMsg function */

/* now check for errors, and so on. */
```

The **Remove()** function is used instead of the **GetMsg()** function to demonstrate that you might have established only one port at which all of your I/O requests will be returned, and you may be checking each request, in turn, with **CheckIO()** to see if it has completed (maybe a disk request, a serial request and a parallel request, all simultaneously outstanding, all using **SendIO()** to transmit their commands to the respective devices).

It is possible that while you are doing other things and checking for completion of I/O, one device may complete its operations and append its message block to your reply port while you are about to check the status of a later-arriving block. If you find that this later one has completed and you call **GetMsg()**, you will remove whichever message is at the head of the list. This message may not necessarily be the one you expect to be removing from the port. **CheckIO()** returns the address of the **IORequest** if the I/O is complete, and you can use this address for the **Remove()** function to remove the correct request block for processing and reuse.

## SECOND ALTERNATIVE MODE FOR READING

Instead of transmitting the read command with either `DoIO()` or `SendIO()`, you might elect to use `BeginIO()`, (the lowest level interface to a device) with the “quick I/O” bit set in the `io_Flags` field.

```
/* same code as in read example, except: */
mySerReq->IOSer.io_Flags = IOF_QUICK; /* use QUICKIO */

BeginIO(mySerReq);
```

The serial device may support quick I/O for certain read requests. As documented in the “Input/Output” chapter in *Amiga ROM Kernel Reference Manual: Exec*, this command may be synchronous or asynchronous. Any write request always clears the quick I/O bit. Various read commands may or may not clear it, depending on whether or not quick I/O occurs.

After executing the code shown above, your program needs to know if the I/O happened synchronously, and it must also test to see if the I/O took place.

```
if((mySerReq->IOSer.io_Flags & IOF_QUICK) == 0)
{
 /* QUICKIO couldn't happen for some reason, so it did it normally...
 * queued the request, cleared the QUICKIO bit, and used the equivalent
 * of SendIO. Might want to have the task doing something else while
 * awaiting the completion * of the I/O. After knowing it is done, must
 * remove the message from the reply port for possible reuse.
 */
 WaitIO(mySerReq);
 /* assumes single-threaded I/O, as compared to
 * the SendIO() example in the previous section */
}
else
{
 /* If flag is still set, IO was synchronous, IORequest was NOT appended
 * to the reply port and there is no need to remove the message from
 * the reply port; continue on with something else.
 */
 ;
}
```

The way you read from the device depends on your need for processing speed. Generally the `BeginIO()` route provides the lowest system overhead when quick I/O is possible. However, if quick I/O did not work, it still requires some overhead for handling of the `IORequest` block.

## TERMINATION OF THE READ

Reading from the serial device can terminate early if an error occurs or if an end-of-file is sensed. You can specify a set of possible end-of-file characters that the serial device is to look for in the input stream. These are contained in an **io\_TermArray** that you provide, using the **SDCMD\_SETPARAMS** command. *Note:* **io\_TermArray** is used only when EOF mode is selected.

If EOF mode is selected, each input data character read into the user's data block is compared against those in **io\_TermArray**. If a match is found, the **IORequest** is terminated as complete, and the count of characters read (including the **TermChar**) is stored in **io\_Actual**. To keep this search overhead as efficient as possible, the serial device requires that the array of characters be in descending order (an example is shown in the summary page in the "Device Summaries" appendix for **SDCMD\_SETPARAMS**). The array has eight bytes and all must be valid (that is, do not pad with zeros unless zero is a valid EOF character).

Fill to the end of the array with the least value **TermChar**. When making an arbitrary choice of EOF character(s), it is advisable to use the lowest value(s) available.

## Writing to the Serial Device

You can write to the serial device as well as read from it. It may be wise to have a separate block for reading and writing to allow simultaneous operation of both reading and writing. The sample code below creates a separate reply port and request for writing to the serial device. Note that it assumes that the **OpenDevice()** function worked properly for the read. It copies the initialized read request block to initialize the write request block. Error-checking has been deliberately left out of this code fragment for brevity but should, of course, be provided in a functional program.

```
/* code fragment to "clone" an existing serial I/O request block instead of
 * opening the device once for read and once for write */

 /* pointer to an existing serial read request block initialized by a
 * call to OpenDevice(SERIALNAME,0,mySerReq,0) */
struct IOExtSer *mySerReq;
LONG i;
BYTE *b,*c;

struct Port *mySerWritePort; /* pointer to a MsgPort at which to receive
 * replies to write requests */
struct IOExtSer *mySerWriteReq; /* pointer to a new request block for serial
 * communications */
```

```

mySerWritePort = CreatePort("mySerialWrite",0);

mySerWriteReq = (struct IOExtSer *)CreateExtIO(mySerWritePort,
 sizeof(struct IOExtSer));

b = (BYTE *)mySerReq; /* start of read request block */
c = (BYTE *)mySerWriteReq; /* start of write request block */

for(i=0; i< sizeof(struct IOExtSer); i++)
 *c++ = *b++;
mySerWriteReq->IOSer.io_Message.mn_ReplyPort = mySerWritePort;
/* clones the request block on a byte by byte basis */
/* Note: it might simply be easier here to have opened the serial device
 * twice. This would reflect the fact that there are two "software entities"
 * that are currently using the device. However, if you are using exclusive
 * access mode, this is not possible and the request block must be copied anyway.
 */

```

Note that this code would require the following clean-up at the termination of the program:

```

cleanupWriteIO:
 DeleteExtIO(mySerWriteReq);
cleanupWritePort:
 DeletePort(mySerWritePort);

```

Now, to perform a write:

```

char dataToWrite[100];
mySerReq->IOSer.io_Data = &dataToWrite[0]; /* where to get the data */
mySerReq->IOSer.io_Length = n; /* write n characters */
mySerReq->IOSer.io_Command = CMD_WRITE; /* say it is a write */
DoIO(mySerReq); /* synchronous request */

```

You can use the `SendIO()` or `BeginIO()` functions as well as `DoIO()`. The same warnings apply as shown above in the discussions about alternative modes of reading.

Note that if `io_Length` is set to -1, the serial device will output your serial buffer until it encounters a value of 0x00 in the data. It transmits this 0 value in addition to the data to match the technique used for serial read shown above. (You can also read data zero-terminated).



## Setting Serial Parameters

You can control the following serial parameters. The parameter name within the serial data structure is shown in table 13-1. All of the fields described in this section are filled in when you call **OpenDevice()** to reflect the current settings of the serial device. Thus, you need not worry about any parameter that you do not need to change.

Table 13-1: Serial Parameters

| Parameter Name     | Characteristic It Controls                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>io_CtlChar</b>  | Control characters to use for xON, xOFF, INQ, ACK respectively. Positioned within an unsigned longword in the sequence from low address to high as listed. INQ and ACK handshaking is not currently supported.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>io_RBufLen</b>  | Size of the buffer that the serial device should allocate for incoming data. Minimum size is 512 bytes. It will not accept a smaller value. This buffer is dynamically allocated by the serial device. If, as you do an <b>SDCMD_SETPARAMS</b> command, it senses a difference between its current value and the value of buffer size you request, it deallocates the old buffer and allocates a new one. Note that it discards all characters that may already be in that old buffer and that you may not have yet had a chance to read. Thus it is wise to make sure that you do not attempt buffer size changes (or any change to the serial device, for that matter) while any I/O is actually taking place. |
| <b>io_ExtFlags</b> | Reserved for future use.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>io_Baud</b>     | The real baud rate you wish to use. A long value from 110 to 292,000. When a value of 110 is requested, it defaults to 112 (the lowest value the hardware can support). Although baud rates above 19,200 are supported by the hardware, software overhead may limit your ability to “catch” every single character that should be received. Output data rate, however, is not software-dependent.                                                                                                                                                                                                                                                                                                                |
| <b>io_BrkTime</b>  | If you issue a break command, this variable specifies how long, in microseconds, the break condition lasts. This value controls the break time for all future break commands until modified by another <b>SDCMD_SETPARAMS</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

- io\_TermArray** A byte-array of eight termination characters, must be in descending order. If EOFMODE is set in the serial flags, this array specifies eight possible choices of character to use as an end of file mark. See the section above titled "Termination of the Read" and the **SDCMD\_SETPARAMS** summary page in the "Device Summaries" appendix for more information.
- io\_ReadLen** How many bits per read character; typically a value of 7 or 8.
- io\_WriteLen** How many bits per write character; typically a value of 7 or 8.
- io\_StopBits** How many stop bits are to be expected when reading a character and to be produced when writing a character; typically 1. A value of 2 is allowed if **io\_WriteLen = 7**.
- io\_SerFlags** Explained below; see "Serial Flags."

| Bit   | Active | Function            |
|-------|--------|---------------------|
| 0     | low    | Busy                |
| 1     | low    | Paper out           |
| 2     | low    | Select              |
| 3     | low    | Data set ready      |
| 4     | low    | Clear to send       |
| 5     | low    | Carrier detect      |
| 6     | low    | Ready to send       |
| 7     | low    | Data terminal ready |
| 8     | high   | Read overrun        |
| 9     | high   | Break sent          |
| 10    | high   | Break received      |
| 11    | high   | Transmit x-OFFed    |
| 12    | high   | Receive x-OFFed     |
| 13-15 | (not)  | (reserved)          |

## SERIAL FLAGS

Table 13-2 shows the flags that can be set to affect the operation of the serial device. Note that the default state of all of these flags is zero.

| Flag Name              | Effect on Device Operation                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>SERB_XDISABLED</b>  | Disable XON-XOFF feature.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>SERB_EOFMODE</b>    | Set this bit if you want the serial device to check I/O characters against <b>io_TermArray</b> and to terminate the <b>IORequest</b> immediately if an end-of-file character has been encountered. <i>Note:</i> This bit can be set and reset directly in the user's <b>IORequest (IOExtSer)</b> block without a call to <b>SDCMD_SETPARAMS</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>SERB_SHARED</b>     | Set this bit if you want to allow other tasks to simultaneously access the serial port. The default is exclusive-access. If someone already has the port, whether for exclusive or shared access, and you ask for exclusive-access, your <b>OpenDevice()</b> call will fail (should be modified only at <b>OpenDevice()</b> ).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>SERB_RAD_BOOGIE</b> | <p>If set, this bit activates high-speed mode. Certain peripheral devices (MIDI, for example) may require high serial throughput. Setting this bit high causes the serial device to skip certain of its internal checking code to speed throughput. In particular, it:</p> <ul style="list-style-type: none"> <li>- Disables parity checking</li> <li>- Bypasses XON/XOFF handling</li> <li>- Uses only 8-bit character length</li> <li>- Will not test for a break signal</li> <li>- Automatically sets <b>SERB_XDISABLED</b> bit</li> </ul> <p>Note that the Amiga is a multitasking system and has immediate processing of software interrupts. If there are other tasks running, it is possible that the serial driver may be unable to keep up with high data transfer rates, even with this bit set.</p> |
| <b>SERB_QUEUEDBRK</b>  | If set, every break command that you transmit will be enqueued. This means that the current serial output commands will be executed in sequence. Then the break command will be executed, all on a FIFO (first in, first out) basis. If this bit is cleared (the default), a break command takes immediate precedence over any serial output already enqueued. When the break command has finished, the interrupted request will continue (if it is not aborted by the user).                                                                                                                                                                                                                                                                                                                                  |

|                       |                                                                                                                                                                                          |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>SERB_7WIRE</b>     | If set (should be established only at <b>OpenDevice()</b> ), the serial device is to use a seven-wire handshaking for RS-232-C communications. Default is three-wire (pins 2, 3, and 7). |
| <b>SERB_PARTY_ODD</b> | If set, selects odd parity. If clear, selects even parity.                                                                                                                               |
| <b>SERB_PARTY_ON</b>  | If set, parity usage and checking is enabled.                                                                                                                                            |

## SETTING THE PARAMETERS

You set the serial parameters by setting the flags and parameters as you desire and then transmitting the command **SDCMD\_SETPARAMS** to the device. Here is an example:

```
mySerReq->IOSer.io_SerFlags &= ~ SERF_PARTY_ODD; /* 'and' with inv#
mySerReq->IOSer.io_SerFlags |= SERF_QUEUEDBRK | SERF_PARTY_ON;
mySerReq->io_BrkTime = 500000; /* 500k microseconds = 1/2 second */
mySerReq->IOSer.io_Command = SDCMD_SETPARAMS;
DoIO(mySerReq); /* synchronous request */
```

The above command would set the bits for queued break and even parity while leaving the other flags unchanged. Notice the difference between the flag names and the flags that you actually set using C. “SERB...” is the name applied to the bit position within the flag word. “SERF...” is the name of a 1 bit in a mask at that bit position.

## Errors from the Serial Device

The possible error returns from the serial device are listed in table 13-3.

Table 13-3: Serial Device Errors

|                                           |    |
|-------------------------------------------|----|
| <code>#define SerErr_DevBusy</code>       | 1  |
| <code>#define SerErr_BaudMismatch</code>  | 2  |
| <code>#define SerErr_InvBaud</code>       | 3  |
| <code>#define SerErr_BufErr</code>        | 4  |
| <code>#define SerErr_InvParam</code>      | 5  |
| <code>#define SerErr_LineErr</code>       | 6  |
| <code>#define SerErr_NotOpen</code>       | 7  |
| <code>#define SerErr_PortReset</code>     | 8  |
| <code>#define SerErr_ParityErr</code>     | 9  |
| <code>#define SerErr_InitErr</code>       | 10 |
| <code>#define SerErr_TimerErr</code>      | 11 |
| <code>#define SerErr_BufOverflow</code>   | 12 |
| <code>#define SerErr_NoDSR</code>         | 13 |
| <code>#define SerErr_NoCTS</code>         | 14 |
| <code>#define SerErr_DetectedBreak</code> | 15 |

## Closing the Serial Device

When the (final, if shared access) **CloseDevice()** is performed, the input buffer is deallocated, the timer device is closed, and the latest parameter settings are saved for the next open.

Typically, you close the serial device with the following function call:

```
CloseDevice(mySerReq);
```

This assumes that the serial device has completed all activities you have requested and has returned all I/O requests to you.

When you have finished with the serial device, it is up to you to deallocate any memory and dependencies you might have used for the serial device communications. If you have used the techniques shown earlier in this chapter to establish the communications in the first place, your clean-up typically will consist of the following code:

```
cleanup2:
 DeleteExtIO(mySerReq,sizeof(struct IOExtSer));
cleanup1:
 DeletePort(mySerPort);
cleanupWriteIO:
 DeleteExtIO(mySerWriteReq);
cleanupWritePort:
 DeletePort(mySerWritePort);
```

## Example Program

Here is an example program that uses static rather than dynamic allocation of the `IOExtSer` request block. It assumes that you have connected a serial terminal device to the Amiga serial port, and it uses the baud rate you have established in Preferences. The program outputs the following status lines to the CLI window:

```
Serial device opened and accepted parameters
Testing character exact-count output thru SendWaitWrite
Test string length of -1 (make system find end of string)
Type 16 characters to send to Amiga...
If no external terminal is attached, waits forever!
```

and outputs the following lines to the external terminal:

```
Device opened ok
User counts characters in string to send, or if null-terminated string, says '-1'
Types 16 characters to send to Amiga
```

At this point, you must type 16 characters on your external terminal. This sample program does not echo characters that you type, so you will not see anything more until all 16 have been typed. Finally the program will respond (to the external terminal) with:

```
You typed these printable characters:
<here it lists the 16 characters>
End of test
54321.....exit
```

Then the program exits, printing "Test completed!" to the CLI window.

```
#include "exec/types.h"
#include "exec/nodes.h"
#include "exec/lists.h"
#include "exec/ports.h"
#include "exec/libraries.h"
#include "exec/devices.h"
#include "exec/io.h"
#include "devices/serial.h"

struct IOExtSer *IORser;
struct MsgPort *port;
char buffer[200];
extern struct MsgPort *CreatePort();
extern struct IORequest *CreateExtIO();

/* Note: to run this program, you must have an external terminal, set
 * at 9600 baud, attached to the Amiga serial port. Additionally the
 * serial.device file must be located in the directory currently
 * assigned to DEVS: (to check this, in AmigaDOS, type: ASSIGN
 * then check the directory (usually the boot CLI disk volume, devs directory.)
 */

main()
{
 int error;
 int actual;
 unsigned long rbl;
 unsigned long brk;
 unsigned long baud;
 unsigned char rwl;
 unsigned char wwl;
 unsigned char sf;
 unsigned long t0;
 unsigned long t1;

 /* SET UP the message port in the I/O request */
 port = CreatePort (SERIALNAME,0);
 if (port == NULL) {
 printf("\nProblems during CreatePort");
 exit(100);
 }

 /* Create the request block for passing info
```

```

 * to and from the serial device. */

IORser = (struct IOExtSer *)CreateExtIO(port,sizeof(struct IOExtSer));
if (IORser == NULL)
{
 printf("\nProblems during CreateExtIO");
 goto cleanup1;
}

open:
/* OPEN the serial device */
if ((error = OpenDevice (SERIALNAME, 0, IORser, 0)) != 0) {
 printf ("Serial device did not open, error = %ld",error);
 goto cleanup1;
}

/* SET PARAMS for the serial device */
rbl = 4096;
rwl = 0x08;
wwl = 0x08;
brk = 750000;
baud= 9600;
sf = 0x00;
t0 = 0x51040303;
t1 = 0x03030303;

if ((error = SetParams (IORser,rbl,rwl,wwl,brk,baud,sf,t0,t1)) != 0) {
 printf ("Set parameters command returned an error: %ld",error);
 goto cleanup2;
}

printf("\nSerial Device opened and accepted parameters");
WriteSer (IORser,"\n\015Device opened ok\n\015", -1);

printf("\nTesting character exact-count output thru SendWaitWrite");
SendWaitWrite (IORser,
 "User counts characters in string to send\n\015", 42);

printf("\nTest string length of -1 (make system find end of string)");
SendWaitWrite (IORser,
 "or if null terminated string, say '-1'\n\015", -1);

printf("\nType 16 characters to send to amiga...");
printf("\nIf no external terminal is attached, waits forever!!");
WriteSer (IORser,

```



```

 "\n\015Type 16 characters to send to amiga\n\015", -1);
actual = ReadSer (IORser,buffer,16);
WriteSer (IORser,
 "\n\015You typed these printable characters:\n\015", -1);
WriteSer (IORser,buffer, actual);
WriteSer (IORser,"\n\015End of test\n\015", -1);
WriteSer (IORser,"54321.....exit\n\015", 16);
printf("\nTest completed!\n");

/* CLOSE the serial.device */
cleanup2:
 CloseDevice (IORser);
cleanup1:
 DeletePort (port);
 exit (0);
}

/* SERIAL I/O functions */

SetParams(io,rbuf_len,rlen,wlen,brk,baud,sf,ta0,ta1)

struct IOExtSer *io;
unsigned long rbuf_len;
unsigned char rlen;
unsigned char wlen;
unsigned long brk;
unsigned long baud;
unsigned char sf;
unsigned long ta0;
unsigned long ta1;

{
 int error;

 io->io_ReadLen = rlen;
 io->io_BrkTime = brk;
 io->io_Baud = baud;
 io->io_WriteLen = wlen;
 io->io_StopBits = 0x01;
 io->io_RBufLen = rbuf_len;
 io->io_SerFlags = sf;
 io->IOSer.io_Command = SDCMD_SETPARAMS;
 io->io_TermArray.TermArray0 = ta0;
 io->io_TermArray.TermArray1 = ta1;

```

```

 if ((error = DoIO (io)) != 0) {
 printf ("serial.device setparams error %ld \n", error);
 }
 return (error);
}

```

```

ULONG ReadSer(io,data,length)
struct IOExtSer *io;
char *data;
ULONG length;
{
 int error;

 io->IOSer.io_Data = (APTR)data;
 io->IOSer.io_Length = length;
 io->IOSer.io_Command = CMD_READ;

 if ((error = DoIO (io)) != 0) {
 printf ("serial.device read error %ld \n", error);
 }
 return (io->IOSer.io_Actual);
}

```

```

WriteSer(io,data,length)
struct IOExtSer *io;
char *data;
int length;
{
 int error;

 io->IOSer.io_Data = (APTR)data;
 io->IOSer.io_Length = length;
 io->IOSer.io_Command = CMD_WRITE;

 if ((error = DoIO (io)) != 0) {
 printf ("serial.device write error %ld \n", error);
 }
 return (error);
}

```

```

ULONG SendWaitWrite(io,data,length)
struct IOExtSer *io;
char *data;
int length;

```

```
{
 int error;

 io->IOSer.io_Data = (APTR)data;
 io->IOSer.io_Length = length;
 io->IOSer.io_Command = CMD_WRITE;

 SendIO (io);

 if ((error = WaitIO (io)) != 0) {
 printf ("serial.device waitio error %ld \n", error);
 }
 return (io->IOSer.io_Actual);
}
```

# Chapter 14

## Parallel Device

This chapter describes software access to the parallel port. The parallel device is accessed via the standard system device access routines and provides some additional functions specifically appropriate to use of this device.

### Introduction

The parallel device can be opened either in exclusive-access or shared mode. Other parallel device parameters can be specified using the PDCMD\_SETPARAMS command after the device has been opened.

## Opening the Parallel Device

Typically, you open the parallel device by using the following function calls:

```
LONG error;
struct Port *myParPort;
struct IOExtPar *myParReq;

/* create a reply port to which parallel
 * device can return the request */
myParPort = CreatePort("myParallel",0);
if(myParPort == NULL) exit(100); /* can't create port? */

/* create a request block appropriate to parallel */
myParReq = (struct IOExtPar *)CreateExtIO(myParPort,
 sizeof(struct IOExtPar));
if(myParReq == NULL) goto cleanup1; /* error during CreateExtIO? */

myParReq->io_ParFlags = 0;
/* accept the default, i.e., exclusive access. Remaining flags all zero,
 * see devices/parallel.h for bit-positions. Definitions included in this
 * chapter. */

error = OpenDevice("parallel.device",0,myParReq,0);
if(error != 0) goto cleanup2; /* device not available? */

...
cleanup2:
 DeleteExtIO(myParReq,sizeof(struct IOExtPar));
cleanup1:
 DeletePort(myParPort);
```

The routines `CreatePort()` and `DeletePort()` are part of `amiga.lib`. Information about the routines `CreateExtIO()` and `DeleteExtIO()` can be found in the appendixes of the *Amiga ROM Kernel Reference Manual: Exec*.

The parallel device is disk-resident. If it has not yet been loaded from disk, it will be read from `DEVS:parallel.device` on the boot AmigaDOS disk. Its parameters will be set up from default values.

During the opening process, the only flag used by the parallel device is the shared/exclusive-access flag. For consistency, however, the other flag bits should be set to zero when the device is opened.

When the parallel device is opened, it opens the timer device and fills the latest parameter settings into the **io\_Request** block. The **OpenDevice()** routine will fill the latest parameter settings into the **io\_Request** block. Note that a parameter change cannot be performed while an I/O request is being processed, because it would invalidate request handling already in progress. Therefore, you must use **PDCMD\_SETPARAMS** only when you have no parallel I/O requests pending.

## Reading from the Parallel Device

You read from the parallel device by sending your **IORequest (IOExtPar)** to the device with a read command. You specify how many bytes are to be transferred and where the data is to be placed. Depending on how you have set your parameters, the request may read the requested number of characters, or it may terminate early.

Here is a sample read command:

```
char myDataArea[100];
myParReq->IOPar.io_Data = &myDataArea[0]; /* where to put the data */
myParReq->IOPar.io_Length = 100; /* read 100 characters */
myParReq->IOPar.io_Command = CMD_READ; /* say it is a read */
DoIO(myParReq); /* synchronous request */
```

If you use this example, your task will be put to sleep waiting until the parallel device reads 100 bytes (or terminates early) and copies them into your read-buffer. Early termination can be caused by error conditions or by the parallel device sensing an end-of-file condition.

Note that the **io\_Length** value, if set to -1, tells the parallel device that you want to read a null-terminated string. The device will read all incoming characters up to and including a byte value of 0x00 in the input stream, then report to you an **io\_Actual** value that is the actual length of the string, excluding the 0 value. Be aware that you must encounter a 0 value in the input stream before the system fills up the buffer you have specified. The **io\_Length** is, for all practical purposes, indefinite. Therefore, you could potentially overwrite system memory if you never encountered the null termination (zero-value byte) in the input stream.

## ALTERNATIVE MODE FOR READING

As an alternative to **DoIO()**, you can use **SendIO()** to transmit the command to the device. In this case, your task can go on to do other things while the parallel device is collecting the bytes for you. You can occasionally do a **CheckIO(myParReq)** to see if the I/O is completed.

```
struct Message *myIO;

/* same code as in above example, except: */
SendIO(myParReq);

 /* do something */
 /* (user code) */
myIO = CheckIO(myParReq);
if(myIO != FALSE) goto ioDone; /* this IO is done */

 /* do something else */
 /* (user code) */
WaitIO(myParReq);
myIO = myParReq; /* if had to wait, need a value for myIO */
}
ioDone:
 Remove(myParPort->mp_MsgList,myIO);
/* use the Remove function rather than the GetMsg function */

/* now check for errors, and so on. */
```

The **Remove()** function is used instead of the **GetMsg()** function to demonstrate that you might have established only one port at which all of your I/O requests will be returned, and you may be checking each request in turn with **CheckIO()** to see if it has completed. These requests could be, for example, a disk request, a parallel request, and a serial request, all simultaneously outstanding and all using **SendIO()** to transmit their commands to the respective devices.

It is possible that while you are doing other things and checking for completion of I/O, one device may complete its operations and append its message block to your reply port when you are about to check the status of a later-arriving block. If you find that this later one has completed and you call **GetMsg()**, you will remove the message at the head of the list. This message may not necessarily be the one you expect to remove from the port. **CheckIO()** returns the address of the **IORequest** if the I/O is complete, and you can use this address for the **Remove()** function to remove the correct request block for processing and reuse.

## TERMINATION OF THE READ

Reading from the parallel device can terminate early if an error occurs or if end of file is sensed. You can specify a set of possible end-of-file characters that the parallel device is to look for in the input stream. These are contained in an **io\_TermArray** that you provide, using the **PDCMD\_SETPARAMS** command. *Note:* **io\_TermArray** is used only when EOF mode is selected.

If EOF mode is selected, each input data character that is read into the user's data block is compared against those in **io\_TermArray**. If a match is found, the **IORequest** is terminated as complete, and the count of characters read (including the **TermChar**) is stored in **io\_Actual**. To keep this search overhead as efficient as possible, the parallel device requires that the array of characters be in descending order (an example is shown in the **PDCMD\_SETPARAMS** summary in the "Device Summaries" appendix. The array has eight bytes and all must be valid (that is, do not pad with zeros unless zero is a valid EOF character). Fill to the end of the array with the least-value **TermChar**. When making an arbitrary choice of EOF character(s), it is advisable to use the lowest value(s) available.

## Writing to the Parallel Device

You can write to the parallel device as well as read from it. It may be wise to have a separate **IORequest** block for reading and writing to allow both operations to take place simultaneously. If you wish to queue multiple commands to the parallel device (either read or write commands), it is acceptable to clone (copy) the I/O request block you receive from the call to **OpenDevice()**. A sample of cloning code is shown in the "Serial Device" chapter.

To perform a write:

```
char dataToWrite[100];
myParReq->IOPar.io_Data = &dataToWrite[0]; /* where to get the data */
myParReq->IOPar.io_Length = n; /* write n characters */
myParReq->IOPar.io_Command = CMD_WRITE; /* say it is a write */
DoIO(myParReq); /* synchronous request */
```

You can use the **SendIO()** or **BeginIO()** functions as well as **DoIO()**. The same warnings apply as shown above in the discussions about alternative modes of reading.

Note that if **io\_Length** is set to -1, the parallel device will output your parallel buffer until it encounters a value of 0x00 in the data. It transmits this 0 value in addition to the data to match the technique used for parallel read shown above. (You can also read data zero-terminated.)



## Setting Parallel Parameters

You can control the parallel parameters shown in table 14-1. The parameter name within the parallel data structure is shown below. All of the fields described in this section are filled in when you call **OpenDevice()** to reflect the current settings of the parallel device. Thus, you need not worry about any parameter that you do not need to change.

Table 14-1: Parallel Parameters

| Parameter Name       | Characteristic It Controls                                                                                                                                                                                                                                                                                                                                              |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>io_PExtFlags</b>  | Reserved for future use.                                                                                                                                                                                                                                                                                                                                                |
| <b>io_PTermArray</b> | A byte-array of eight termination characters, must be in descending order. If EOFMODE is set in the parallel flags, this array specifies eight possible choices of character to use as an end-of-file mark. See the <b>PDCMD_SETPARAMS</b> summary page in the "Device Summaries" appendix and the section above titled "Termination of the Read" for more information. |
| <b>io_ParFlags</b>   | Explained below; see "Parallel Flags."                                                                                                                                                                                                                                                                                                                                  |

### PARALLEL FLAGS

The flags shown in table 14-2 can be set to affect the operation of the parallel device. Note that the default state of all of these flags is zero.

Table 14-2: Parallel Flags

| Flag Name           | Effect on Device Operation                                                                                                                                                                                                                                                                                                                       |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>PARB_EOFMODE</b> | Set this bit if you want the parallel device to check I/O characters against <b>io_TermArray</b> and terminate the <b>IORequest</b> immediately if an end-of-file character has been encountered. <i>Note:</i> This bit can be set and reset directly in the user's <b>IORequest (IOExtPar)</b> block without a call to <b>PDCMD_SETPARAMS</b> . |
| <b>PARB_SHARED</b>  | Set this bit if you want to allow other tasks to simultaneously access the parallel port. The default is exclusive access. If someone already has the port, whether for exclusive or shared access, and you ask for exclusive access, your <b>OpenDevice()</b> call will fail (should be modified only at <b>OpenDevice()</b> ).                 |

## SETTING THE PARAMETERS

You set the parallel parameters by setting the flags and parameters as you desire and then transmitting the command **PDCMD\_SETPARAMS** to the device. Here is an example:

```
myParReq->IOPar.io_ParFlags &= ~ PARF_EOFMODE;
/* "and" with inverse */
myParReq->IOPar.io_Command = PDCMD_SETPARAMS;
DoIO(myParReq); /* synchronous request */
```

The above command would cancel EOFMODE (use of the **io\_TermArray**), leaving the other flags unchanged. Notice the difference between the flag names and the flags that you actually set using C. "PARB..." is the name applied to the bit position within the flag word. "PARF..." is the name of a 1 bit in a mask at that bit position.

## Errors from the Parallel Device

The possible error returns from the parallel device are listed in table 14-3.

Table 14-3: Parallel Device Errors

```
#define ParErr_DevBusy 1
#define ParErr_BufToBig 2
#define ParErr_InvParam 3
#define ParErr_LineErr 4
#define ParErr_NotOpen 5
#define ParErr_PortReset 6
#define ParErr_InitErr 7
```

## Closing the Parallel Device

When the (final, if shared access) **CloseDevice()** is performed, the timer device is closed, and the latest parameter settings are saved for the next open.

Typically, you close the parallel device with the following function call:

```
CloseDevice(myParReq);
```

This assumes that the parallel device has completed all activities you have requested and has returned all I/O requests to you. When you have finished with the parallel device, it is up to you to deallocate any memory and dependencies you might have used for the parallel device communications. If you have used the techniques shown earlier in this chapter to establish the communications in the first place, your clean-up typically will consist of the following code:

```
cleanup2:
 DeleteExtIO(myParReq,sizeof(struct IOExtPar));
cleanup1:
 DeletePort(myParPort);
cleanupWriteIO:
 DeleteExtIO(myParWriteReq);
cleanupWritePort:
 DeletePort(myParWritePort);
```

## Example Program

Here is an example program that uses static rather than dynamic allocation of the **IOExtPar** request block. It assumes that you have connected a parallel I/O device to the Amiga parallel port.

```
#include "exec/types.h"
#include "exec/nodes.h"
#include "exec/lists.h"
#include "exec/ports.h"
#include "exec/libraries.h"
#include "exec/devices.h"
#include "exec/io.h"
#include "devices/parallel.h"

struct IOExtPar IORpar;
struct MsgPort *port;
char buffer[64000];
extern struct MsgPort *CreatePort();

main()
{
 int error;
 int actual;
 unsigned char pflags;
 unsigned long pt0;
 unsigned long pt1;

open:
 /* OPEN the parallel.device */
 if ((error = OpenDevice (PARALLELNAME, 0, &IORpar, 0)) != 0) {
 printf ("bad news %ld on Open \n", error);
 exit (error);
 }

 /* SET UP the message port in the I/O request */
 port = CreatePort (PARALLELNAME,0);
 IORpar.IOPar.io_Message.mn_ReplyPort = port;

 /* SET PARAMS for the parallel.device */
 pflags = PARF_EOFMODE;
 pt0 = 0x51040303;
 pt1 = 0x03030303;
```

```

if ((error = setparams (pflags,pt0,pt1)) != 0) {
 printf ("bad news %ld on setup \n", error);
 DeletePort();
 exit (error);
}

actual = readPar (buffer,60000);

/* CLOSE the parallel.device */
CloseDevice (&IORpar);
DeletePort (port);
exit (0);
}

/* PARALLEL I/O functions */

setparams(pf,ta0,ta1)

 unsigned char pf;
 unsigned long ta0;
 unsigned long ta1;

{
 int error;

 IORpar.io_ParFlags = pf;
 IORpar.IOPar.io_Command = PDCMD_SETPARAMS;
 IORpar.io_PTermArray.PTermArray0 = ta0;
 IORpar.io_PTermArray.PTermArray1 = ta1;

 if ((error = DoIO (&IORpar)) != 0) {
 printf ("parallel.device setparams error %ld \n", error);
 }
 return (error);
}

readPar(data,length)
 char *data;
 ULONG length;
{
 int error;

 IORpar.IOPar.io_Data = data;
 IORpar.IOPar.io_Length = length;
 IORpar.IOPar.io_Command = CMD_READ;

```

```

if ((error = DoIO (&IORpar)) != 0) {
 printf ("parallel.device read error %ld \n", error);
}
return (IORpar.IOPar.io_Actual);
}

writePar(data,length)
char *data;
int length;
{
 int error;

 IORpar.IOPar.io_Data = data;
 IORpar.IOPar.io_Length = length;
 IORpar.IOPar.io_Command = CMD_WRITE;

 if ((error = DoIO (&IORpar)) != 0) {
 printf ("parallel.device write error %ld \n", error);
 }
 return (error);
}

```



# Chapter 15

## Printer Device

### Introduction

There are four basic ways of doing output to a printer on the Amiga computer and three basic kinds of output you can send. You can send your output to these devices:

- o **PRT:**—the DOS printer device



- o **SER:**— the DOS serial device
- o **PAR:**— the DOS parallel device
- o **printer.device**— to directly access the printer device itself

Your output can take the following form:

- o A character stream, consisting of commands and data (if sent through DOS or directly to the printer device)
- o A command (if sent directly to the printer device)
- o A graphics dump (also sent directly to the printer device)

The following section explains the various possible access pathways to the printer itself, along with the advantages and disadvantages of each pathway.

### **PRT:— THE AMIGADOS PRINTER DEVICE**

**PRT:** is the AmigaDOS printer device. By using the Workbench Preferences tool, you can direct the output to either a serial or parallel printer, which is the generic printer configured on the system. You may print (output) escape sequences to **PRT:** to specify the options you want. The escape sequences you send are interpreted by the printer driver and (usually different) escape sequences are forwarded to the printer. This is by far the easiest method for most applications. **PRT:** may be opened just like any other AmigaDOS file.

### **SER:— THE AMIGADOS SERIAL DEVICE**

**SER:** is the AmigaDOS serial device. If you “know” that the printer is connected to the serial port (you should not) and you “know” what kind of printer it is (again, you should not) then you could use AmigaDOS to open **SER:** and output characters to it, causing it to print. *This practice is strongly discouraged!* Characters you send are not examined or converted.

### **PAR:— THE AMIGADOS PARALLEL DEVICE**

**PAR:** is the AmigaDOS parallel device. The warnings given in the paragraph above apply here as well.

## THE PRINTER DEVICE

By opening the Exec printer device directly, you have full control over the printer. You can either send escape sequences as shown in the command definitions table below for printer control or call the **RawWrite()** routine to send raw characters directly to your printer with no processing at all. Using this technique would be similar to sending raw characters to **SER:** or **PAR:** from AmigaDOS (but you do not need to know which one is connected to the printer). Also note that all "commands" to the printer transmitted through the DOS printer access path must take the form of a character stream. Direct access to the printer device allows you to transmit other commands, such as reset or flush or, for graphics dumps, **DumpRPort()** (dump a raster to a graphics-capable printer).

## Printer Device Output

The printer device can be thought of as kind of a filter, in that some printers respond in one way to a command output and some respond in another. The printer device, as a standard printer interface, recognizes command sequences. Depending on the printer-dependent configuration that is currently loaded (by the Preferences tool), the printer device either ignores the command sequences or perhaps translates them into an entirely different sequence that this printer can actually understand and obey.

## Opening the AmigaDOS Printer Device

You can open the DOS printer device just as though it were a normal DOS output file. Here is an example program segment that accomplishes this:

```
struct File *file;

file = Open("PRT:", MODE_NEWFILE);
if (file == 0) exit(PRINTER_WONT_OPEN);
```

Then, to print use code like this:

```
actual_length = Write(file, dataLocation, length);
```

where

**file**

is a file handle (see the *AmigaDOS Developers Manual*).

**dataLocation**

is a pointer to the first character in the output stream you wish to write.

**length**

is the length of the output stream.

**actual\_length**

is the actual length of the write. For the printer device, if there are no errors, this is likely to always be the same as the length of write requested. The only exception is if you specify a value of -1 for length. In this case, -1 for length means that a null (0) terminated stream is being written to the printer device. The device returns the count of characters written prior to encountering the null. If it returns a value of -1 as **actual\_length**, there has been an error.

Note that the **Open()** function could be called with **SER:** or **PAR:** if you do not want to have any character translation performed during the printer I/O. When the printer I/O is complete, and your program is ready to exit, you should close the device. Here is a sample function call that you could use:

```
Close(file);
```

Note that printer I/O through the DOS versions of the printer device must be done by a process, not by a task. DOS utilizes information in the process control block and would become confused if a simple task attempted to perform these activities. Printer I/O using the printer device directly, however, *can* be performed by a task.

## Data Structures Used During Printer I/O

This section shows you how to set up for Exec printer I/O. There are three distinct kinds of data structures required by the printer I/O routines. Some of the printer commands, such as start, stop, and flush, require only an **IStdReq**. Others, such as write, require a larger data structure called an **IODRPreq** (for “dump a RastPort”) or **IOPrtCmdReq** (for “printer command request”). For convenience, the printer device has defined a single data structure, called **printerIO**, that can be used to represent any of the three different kinds of printer communications request blocks.

The data structure type **printerIO** used in the following examples is a C-language union defined as:

```

union printerIO{
struct IOStdReq ios;
struct IODRPRReq iodrp;
struct IOPrtCmdReq iopc;
}

```

This means that one memory area can be used to represent three distinct forms of memory layout for the three different types of data structures that must be used to pass commands to the printer device. Some of the commands are simple and can use an **IOStdReq**. Some of the commands require many more parameters and extend the basic I/O request block accordingly. If you use the function **CreateExtIO()**, you can automatically allocate enough memory to hold the largest structure in the union statement.

## Creating an I/O Request

Printer I/O, like the I/O of other devices, requires that you create an I/O request message that you pass to the printer device for processing. The message contains the command as well as a data area. For a write, there will be a pointer in the data area to the stream of information you wish to write to the printer.

The following program fragment can be used to create the message block that you use for printer communications.

```

union printerIO *printerMsg; /* I/O request block pointer */
struct Port *printerPort; /* a port at which to receive */

printerPort = CreatePort("my.print.port",0);
printerMsg = (union printerIO *)CreateExtIO(printerPort,
 sizeof(union printerIO));

```

Error handling is not shown here. It is deferred to the example at the end of the chapter.

The routine **CreatePort()**, which is part of *amiga.lib*, and the routine **CreateExtIO()** may be found in the appendixes of the *Amiga ROM Kernel Reference Manual: Exec*.

Note that there are two additional kinds of I/O request blocks that, for some commands, must be prepared for sending to the printer. They are called **IODRPRReq** and **IOPrtCmdReq**. Both are outlined in the include file *devices/printer.h*. The function call to **CreateExtIO()** returns a pointer to a memory block the size of the largest form of printer **IORrequest**.

## Opening a Printer Device

You open a path to the printer device using code like the following:

```
int
OpenPrinter(request)
union printerIO *request;
{
 return(OpenDevice("printer.device",0,request,0));
}
```

This routine returns a value of zero if the printer device was opened successfully and a value other than zero if it did not open.

## Writing to the Printer

There are three forms of writing to the printer. The first uses a character stream that you create, possibly containing escape sequences to be processed by the printer driver ("PrintString" example) or containing just about anything else that is to be passed directly to the printer ("PrintRaw" example). The second form of write passes a command to the printer ("PrintCmd" example). The third form asks for a graphics dump of a drawing area ("Printer-Dump" example).

To write to the printer, you pass to the printer device the system standard command `CMD_WRITE`. Here are routines that can be used to send this command:

```
/* Send a NULL-terminated string to the printer */

/* Assumes printer device is open and printerMsg is correctly initialized.
 * Watches for embedded "escape-sequences" and handles them as defined.
 */

int
PrintString(request,string)
union printerIO *request;
char *string;
{
 request->ios.io_Command = CMD_WRITE;
 request->ios.io_Data = string;
 request->ios.io_Length = -1;
 /* if -1, the printer assumes it has been given
```

```

 * a null-terminated string.
 */
return(DoIO(request));
}

/* Send RAW character stream to the printer directly,
 * avoid "escape-sequence" parsing by the device.
 */

int
PrintRaw(request,buffer,count)
union printerIO *request; /* a properly initialized request block */
char *buffer; /* where is the output stream of characters */
int count; /* how many characters to output */
{
 /* queue a printer raw write */
 request->ios.io_Command = PRD_RAWWRITE;
 request->ios.io_Data = buffer;
 request->ios.io_Length = count;
 return(DoIO(request));
}

```

## PRINTER COMMAND DEFINITIONS

The following table describes the supported printer functions. You can use the escape sequences with **PRT:** and the printer device.

To transmit a command to the printer device, you can either formulate a character stream containing the material shown in the "Escape Sequence" column of table 15-1 below or send an **IORequest** to the printer device specifying which of these commands you wish to have performed. A sample routine for transmitting commands is shown immediately following the command table.

Again, recall that **SER:** and **PAR:** will ignore all of these and pass them directly on to the attached device.

Table 15-1: Printer Device Command Functions

| Name    | Cmd No. | Escape Sequence | Function                                                                                              | Defined by:   |
|---------|---------|-----------------|-------------------------------------------------------------------------------------------------------|---------------|
| aRIS    | 0       | ESCc            | Reset                                                                                                 | ISO           |
| aRIN    | 1       | ESC#1           | Initialize                                                                                            | +++           |
| aIND    | 2       | ESCD            | Lf                                                                                                    | ISO           |
| aNEL    | 3       | ESCE            | Return,lf                                                                                             | ISO           |
| aRI     | 4       | ESCM            | Reverse lf                                                                                            | ISO           |
| aSGR0   | 5       | ESC[0m          | Normal char set                                                                                       | ISO           |
| aSGR3   | 6       | ESC[3m          | Italics on                                                                                            | ISO           |
| aSGR23  | 7       | ESC[23m         | Italics off                                                                                           | ISO           |
| aSGR4   | 8       | ESC[4m          | Underline on                                                                                          | ISO           |
| aSGR24  | 9       | ESC[24m         | Underline off                                                                                         | ISO           |
| aSGR1   | 10      | ESC[1m          | Boldface on                                                                                           | ISO           |
| aSGR22  | 11      | ESC[22m         | Boldface off                                                                                          | ISO           |
| aSFC    | 12      | ESC[nm          | Set foreground color<br>where n stands for a pair<br>of ASCII digits, 3 followed<br>by any number 0-9 | ISO           |
| aSBC    | 13      | ESC[nm          | Set background color<br>Where n stands for<br>a pair of ASCII digits, 4<br>followed by any number 0-9 | ISO           |
| aSHORP0 | 14      | ESC[0w          | Normal pitch                                                                                          | DEC           |
| aSHORP2 | 15      | ESC[2w          | Elite on                                                                                              | DEC           |
| aSHORP1 | 16      | ESC[1w          | Elite off                                                                                             | DEC           |
| aSHORP4 | 17      | ESC[4w          | Condensed fine on                                                                                     | DEC           |
| aSHORP3 | 18      | ESC[3w          | Condensed off                                                                                         | DEC           |
| aSHORP6 | 19      | ESC[6w          | Enlarged on                                                                                           | DEC           |
| aSHORP5 | 20      | ESC[5w          | Enlarged off                                                                                          | DEC           |
| aDEN6   | 21      | ESC[6"z         | Shadow print on                                                                                       | DEC (sort of) |
| aDEN5   | 22      | ESC[5"z         | Shadow print off                                                                                      | DEC           |
| aDEN4   | 23      | ESC[4"z         | Doublestrike on                                                                                       | DEC           |
| aDEN3   | 24      | ESC[3"z         | Doublestrike off                                                                                      | DEC           |
| aDEN2   | 25      | ESC[2"z         | NLQ on                                                                                                | DEC           |
| aDEN1   | 26      | ESC[1"z         | NLQ off                                                                                               | DEC           |
| aSUS2   | 27      | ESC[2v          | Superscript on                                                                                        | +++           |
| aSUS1   | 28      | ESC[1v          | Superscript off                                                                                       | +++           |
| aSUS4   | 29      | ESC[4v          | Subscript on                                                                                          | +++           |
| aSUS3   | 30      | ESC[3v          | Subscript off                                                                                         | +++           |
| aSUS0   | 31      | ESC[0v          | Normalize the line                                                                                    | +++           |
| aPLU    | 32      | ESCL            | Partial line up                                                                                       | ISO           |

|         |    |          |                         |               |
|---------|----|----------|-------------------------|---------------|
| aPLD    | 33 | ESCK     | Partial line down       | ISO           |
| aFNT0   | 34 | ESC(B    | US char set             | DEC           |
| aFNT1   | 35 | ESC(R    | French char set         | DEC           |
| aFNT2   | 36 | ESC(K    | German char set         | DEC           |
| aFNT3   | 37 | ESC(A    | UK char set             | DEC           |
| aFNT4   | 38 | ESC(E    | Danish I char set       | DEC           |
| aFNT5   | 39 | ESC(H    | Swedish char set        | DEC           |
| aFNT6   | 40 | ESC(Y    | Italian char set        | DEC           |
| aFNT7   | 41 | ESC(Z    | Spanish char set        | DEC           |
| aFNT8   | 42 | ESC(J    | Japanese char set       | +++           |
| aFNT9   | 43 | ESC(6    | Norwegian char set      | DEC           |
| aFNT10  | 44 | ESC(C    | Danish II char set      | +++           |
| aPROP2  | 45 | ESC 2p   | Proportional on         | +++           |
| aPROP1  | 46 | ESC 1p   | Proportional off        | +++           |
| aPROP0  | 47 | ESC 0p   | Proportional clear      | +++           |
| aTSS    | 48 | ESC n E  | Set proportional offset | ISO           |
| aJFY5   | 49 | ESC 5 F  | Auto left justify       | ISO           |
| aJFY7   | 50 | ESC 7 F  | Auto right justify      | ISO           |
| aJFY6   | 51 | ESC 6 F  | Auto full justify       | ISO           |
| aJFY0   | 52 | ESC 0 F  | Auto justify off        | ISO           |
| aJFY3   | 53 | ESC 3 F  | Letter space (justify)  | ISO (special) |
| aJFY1   | 54 | ESC 1 F  | Word fill(auto center)  | ISO (special) |
| aVERP0  | 55 | ESC 0z   | 1/8" line spacing       | +++           |
| aVERP1  | 56 | ESC 1z   | 1/6" line spacing       | +++           |
| aSLPP   | 57 | ESC nt   | Set form length n       | DEC           |
| aPERF   | 58 | ESC nq   | Perf skip n (n>0)       | +++           |
| aPERF0  | 59 | ESC 0q   | Perf skip off           | +++           |
| aLMS    | 60 | ESC#9    | Left margin set         | +++           |
| aRMS    | 61 | ESC#0    | Right margin set        | +++           |
| aTMS    | 62 | ESC#8    | Top margin set          | +++           |
| aBMS    | 63 | ESC#2    | Bottom margin set       | +++           |
| aSTBM   | 64 | ESC n;nr | T&B margins             | DEC           |
| aSLRM   | 65 | ESC n;ns | L&R margin              | DEC           |
| aCAM    | 66 | ESC#3    | Clear margins           | +++           |
| aHTS    | 67 | ESCH     | Set horiz tab           | ISO           |
| aVTS    | 68 | ESCJ     | Set vertical tabs       | ISO           |
| aTBC0   | 69 | ESC 0g   | Clr horiz tab           | ISO           |
| aTBC3   | 70 | ESC 3g   | Clear all h tab         | ISO           |
| aTBC1   | 71 | ESC 1g   | Clr vertical tabs       | ISO           |
| aTBC4   | 72 | ESC 4g   | Clr all v tabs          | ISO           |
| aTBCALL | 73 | ESC#4    | Clr all h & v tabs      | +++           |
| aTBSALL | 74 | ESC#5    | Set default tabs        | +++           |
| aEXTEND | 75 | ESC n"x  | Extended commands       | +++           |



Legend:

- ISO indicates that the sequence has been defined by the International Standards Organization. This is also very similar to ANSI x3.64.
- DEC indicates a control sequence defined by Digital Equipment Corporation.
- +++ indicates a sequence unique to Amiga.
- n stands for a decimal number expressed as a set of ASCII digits, for example 12.

## Transmitting a Command to the Printer Device

As noted above, to transmit a command to the printer device, you can either formulate an escape sequence and send it via the **CMD\_WRITE** command, or you can utilize the command names and pass parameters and the command to the device. Here is a sample routine that uses the system command **PRD\_PRTCOMMAND** to transmit a command to the device:

```
int
PrintCommand(request,command, p0, p1, p2, p3)
union printerIO *request;
int command, p0, p1, p2, p3; /* command and its parameters */
{
 /* queue a printer command */
 request->iopc.io_Command = PRD_PRTCOMMAND;
 request->iopc.io_PrtCommand = command;
 request->iopc.io_Parm0 = p0;
 request->iopc.io_Parm1 = p1;
 request->iopc.io_Parm2 = p2;
 request->iopc.io_Parm3 = p3;
 return(DoIO(request));
}
```

As an example, suppose you wanted to set the left and right margins on your printer to columns 1 and 79 respectively. Here is a sample call to the **PrintCommand()** function for this purpose:

```
PrintCommand(aSLRM, 1, 79, 0, 0);
```

Consult the function table. Wherever there is a value of “n” to be substituted, it will be utilized from the next available parameter for this command. Most of the commands in the table need no parameters; some need one and others need two. Few, if any, require more than two parameters; however, this function provides room for expansion.

## Dumping a RastPort to the Printer

You can dump a **RastPort** (drawing area) to the printer by sending the command **PRD\_DUMPRPORT** to the printer, along with several parameters that define how the dump is to be accomplished. The parameters shown in the sample dump function below are completely described in the summary for **DumpRPort()** in the “Device Summaries” appendix.

```
int
DumpRPort(request,rastPort, colorMap, modes, sx,sy, sw,sh, dc,dr, s)
 union printerIO *request;
 struct RastPort *rastPort;
 struct ColorMap *colorMap;
 ULONG modes;
 UWORD sx, sy, sw, sh;
 LONG dc, dr;
 UWORD s;
 {
 request->iodrp.io_Command = PRD_DUMPRPORT;
 request->iodrp.io_RastPort = rastPort;
 request->iodrp.io_ColorMap = colorMap;
 request->iodrp.io_Modes = modes;
 request->iodrp.io_SrcX = sx;
 request->iodrp.io_SrcY = sy;
 request->iodrp.io_SrcWidth = sw;
 request->iodrp.io_SrcHeight = sh;
 request->iodrp.io_DestCols = dc;
 request->iodrp.io_DestRows = dr;
 request->iodrp.io_Special = s;
 return(DoIO(request));
 }
```

As an example of this function, suppose you wanted to dump the current contents of the Workbench screen to the printer. The typical program code shown below would accomplish it. Note that during the dump no other tasks should be writing to the screen, nor should you use the mouse to move windows or otherwise modify the screen appearance.

```

/*
 * Author: Rob Peck, 12/1/85
 * Modified: Carolyn Scheppner, 04/08/86
 *
 * This code may be freely utilized to develop programs for the Amiga.
 */

#include "exec/types.h"
#include "intuition/intuition.h"
#include "devices/printer.h"
#define INTUITION_WONT_OPEN 1000

union printerIO {
 struct IOStdReq ios;
 struct IODRPReq iodrp;
 struct IOPrtCmdReq iopc;
};

union printerIO *request; /* a pointer to a request block */

extern int DumpRPort();
extern struct IORequest *CreateExtIO();
extern struct MsgPort *CreatePort();

struct IntuitionBase *IntuitionBase;

main()
{
 struct Screen *screen;
 struct RastPort *rp;
 struct ViewPort *vp;
 struct ColorMap *cm;
 struct MsgPort *printerPort; /* at which to receive reply */
 int modes,width,height,error;

 IntuitionBase = (struct IntuitionBase *)OpenLibrary(
 "intuition.library", 0);
 if (IntuitionBase == NULL) exit(INTUITION_WONT_OPEN);

 screen = IntuitionBase->FirstScreen; /* ptr to front Screen */

 vp = &screen->ViewPort; /* get screen's ViewPort, from
 * which the ColorMap will be gotten */
 rp = &screen->RastPort; /* get screen's RastPort, which
 * is what gets dumped to printer */

```

```

cm = vp->ColorMap; /* retrieve pointer to colormap for
 * the printer dump */
modes = vp->Modes; /* retrieve the modes variable */
width = screen->Width; /* retrieve width and */
height = screen->Height; /* height to print */

printerPort = CreatePort("my.print.port",0);
request = (union printerIO *)CreateExtIO(printerPort,
 sizeof(union printerIO));

error = OpenPrinter(request);
if(error != 0) goto cleanup2;

Delay(300); /* 300/60 = 6 seconds delay before it starts */
error = DumpRPort(
 request, /* pointer to initialized request */
 rp, /* RastPort pointer */
 cm, /* color map pointer */
 modes, /* low, high res, etc (display modes)*/
 0, 0, /* x and y offsets into rastport */
 width,height, /* source size */
 0,0, /* dest size 0 because of Special */
 SPECIAL_FULLCOLS | SPECIAL_ASPECT /* Special */
 /* Special = print max width */
 /* with proportional height */
);
ClosePrinter(request);

cleanup2:
DeleteExtIO(request, sizeof(union printerIO));
DeletePort(printerPort);
cleanup1:
CloseLibrary(IntuitionBase);

} /* end of demo screen dump */

/*****
/* printersupport.c rtns
*****/

/* OPEN THE PRINTER */
int
OpenPrinter(request)

```

```

 union printerIO *request;
 {
 return(OpenDevice("printer.device",0,request,0));
 }

/* CLOSE THE PRINTER */
int
ClosePrinter(request)
 union printerIO *request;
 {
 CloseDevice(request);
 return(0);
 }

/* Send a null-terminated string to the printer. Assumes printer device
 * is open and printerMsg is correctly initialized. Watches for embedded
 * "escape-sequences" and handles them as defined.
 */

int
PrintString(request,string)
 union printerIO *request;
 char *string;
 {
 request->ios.io_Command = CMD_WRITE;
 request->ios.io_Data = (APTR)string;
 request->ios.io_Length = -1;
 /* if -1, the printer assumes it has been given a null terminated string. */
 return(DoIO(request));
 }

/* Send RAW character stream to the printer directly,
 * avoid "escape-sequence" parsing by the device.
 */

int
PrintRaw(request,buffer,count)
 union printerIO *request; /* a properly initialized request block */
 char *buffer; /* where is the output stream of characters */
 int count; /* how many characters to output */
 {
 /* queue a printer raw write */
 request->ios.io_Command = PRD_RAWWRITE;
 request->ios.io_Data = (APTR)buffer;
 }

```

```

request->ios.io_Length = count;

return(DoIO(request));
}

/* Send Printer Command */
int
PrintCommand(request,command, p0, p1, p2, p3)
union printerIO *request;
int command, p0, p1, p2, p3; /* command and its parameters */
{
/* queue a printer command */
request->iopc.io_Command = PRD_PRTCOMMAND;
request->iopc.io_PrtCommand = command;
request->iopc.io_Parm0 = p0;
request->iopc.io_Parm1 = p1;
request->iopc.io_Parm2 = p2;
request->iopc.io_Parm3 = p3;
return(DoIO(request));
}

```

## ADDITIONAL NOTES ABOUT GRAPHICS DUMPS

The print command accepts a “use the largest area you have” specification that looks at the Preferences active print width and active print height to bound the size of the print. These values are specified as a character count and a character size specification. Thus, the width of the print is bounded by the number of inches specified by the following equation:  $(\text{RIGHT\_MARGIN} - \text{LEFT\_MARGIN} + 1) / \text{CHARACTERS\_PER\_INCH}$ . The height is specified by the equation:  $\text{LENGTH} / \text{LINES\_PER\_INCH}$ .

**NumRows** in the printer tag refers to the number of dots in the graphics print element, and can be used by graphics render code to determine how much buffer space is needed to compose a line of graphics output. It has not been used in practice; the number has instead been hard coded into the render function specific to the printer.

If the printer for which you are developing can be set to unidirectional mode under software control, we recommend that you put this in the initialization code for the printer (see case 0 Master Initialization, below). This produces better-looking printouts and under most conditions (believe it or not) a faster printout.

## Creating a Printer Driver

Creating a printer-dependent code fragment for the printer device involves writing the data structures and code, compiling and assembling it, and linking it to produce an Amiga object binary file. The first piece in that file is the **PrinterSegment** structure described in *devices/prtbase.h* and *devices/prtbase.i* (which is pointed to by the BPTR returned by the **LoadSeg()** of the object file).

You specify the printer-dependent object file to load by specifying “custom printer” in Preferences and filling in the custom printer name with the name of the object file (relative to the directory *DEVS:printers/*).

The printer-dependent code **PrinterSegment** contains the **PrinterExtendedData** (PED) structure (also described in *devices/prtbase.h* and *devices/prtbase.i* at the beginning of the object). The PED structure contains data describing the capabilities of the printer, as well as pointers to code and other data. Here is the assembly code for a sample **PrinterSegment**, which would be linked to the beginning of the sequence of files describing the printer-dependent code fragment.

```

*
* printer device dependent code tag
*

 ; named sections are easier to exactly place in the linked file
 SECTION custom

*----- Included Files -----

 INCLUDE "exec/types.i"
 INCLUDE "exec/nodes.i"

 INCLUDE "revision.i" ; contains VERSION & REVISION

 INCLUDE "devices/prtbase.i"

*----- Imported Names -----

 XREF _Init
 XREF _Expunge
 XREF _Open
 XREF _Close
 XREF _CommandTable
 XREF _DoSpecial
```

XREF        \_Render

\*----- Exported Names -----

XDEF        \_PEDData

\*\*\*\*\*

```
 ; in case anyone tries to execute this
MOVEQ #0,D0
RTS
```

```
DC.W VERSION
DC.W REVISION
```

\_PEDData:

```
DC.L printerName
DC.L _Init
DC.L _Expunge
DC.L _Open
DC.L _Close
DC.B PPC_BWGFY ; PrinterClass
DC.B PCC_BW ; ColorClass
DC.B 80 ; MaxColumns
DC.B 1 ; NumCharSets
DC.W 8 ; NumRows
DC.L 960 ; MaxXDots
DC.L 0 ; MaxYDots
DC.W 120 ; XDotsInch
DC.W 82 ; YDotsInch
DC.L _CommandTable ; Command Strings
DC.L _DoSpecial ; Command Code
DC.L _Render ; Graphics Render
DC.L 30 ; Timeout
```

printerName:

```
DC.B 'Custom Printer Name'
DC.B 0
EVEN
```

The printer name should be the brand name of the printer that is available for use by programs wishing to be specific about the printer name in any diagnostic or instruction messages. The four functions at the top of the structure are used to initialize this printer-dependent code:

```
((*PED->ped_Init))(PD);
```

This is called when the printer-dependent code is loaded and provides a pointer to the



printer device for use by the printer-dependent code. It can also be used to open up any libraries or devices needed by the printer-dependent code.

**(\*PED->ped\_Expunge)();**

This is called immediately before the printer-dependent code is unloaded, to allow it to close any resources obtained at initialization time.

**(\*PED->ped\_Open)(ior);**

This is called in the process of an **OpenDevice()** call, after the Preferences are read and the correct primitive I/O device (parallel or serial) is opened. It must return zero if the open is successful, or nonzero to terminate the open and return an error to the user.

**(\*PED->ped\_Close)(ior);**

This is called in the process of a **CloseDevice()** call to allow the printer-dependent code to close any resources obtained at open time.

The **pd\_** variable provided as a parameter to the initialization call is a pointer to the **PrinterData** structure described in *devices/prtbase.h* and *devices/prtbase.i*. This is also the same as the **io\_Device** entry in printer I/O requests.

#### **pd\_SegmentData**

This points back to the **PrinterSegment**, which contains the PED.

#### **pd\_PrintBuf**

This is available for use by the printer-dependent code—it is not otherwise used by the printer device.

**(\*pd\_PWrite)(data, length);**

This is the interface routine to the primitive I/O device. This routine uses two I/O requests to the primitive device, so writes are double-buffered. The data parameter points to the byte data to send, and the length is the number of bytes.

**(\*pd\_PBothReady)();**

This waits for both primitive I/O requests to complete. This is useful if your code does not want to use double buffering. If you want to use the same data buffer for successive **pd\_PWrites**, you must separate them with a call to this routine.

#### **pd\_Preferences**

This is the copy of Preferences in use by the printer device, obtained when the printer was opened.

The timeout field is the number of seconds that an I/O request from the printer device will remain posted and unsatisfied to the primitive I/O device (parallel or serial) before the timeout requester is presented to the user. This value should be large enough to avoid the requester during normal printing.

## SAMPLE CODE

To help you in developing custom printer drivers for the Amiga, four sets of source files have been included as a part of this document. The files include *init.asm*, *printertag.asm*, *data.c*, *render.c*, and *dospecial.c*.

Four sets of files for four different types of printers are provided:

- diablo\_c - an example of a ymcb color printer
- epson - an example of a b/w printer
- okimate20 - an example of a ymc\_bw printer (has two render.c functions)
- hpplus - an example of a single-sheet, multiple-density printer

The source files for the hpplus includes one additional C-language source, named *density.c*.

In addition, you will also need certain files that are common to all printer drivers. These are called *macros.i* and are printer assembly code macros that *init.asm* uses. All of these files are in the "Printer Device Source Code" appendix of this manual.

## WRITING A GRAPHICS PRINTER DRIVER

Designing the graphics portion of a custom printer driver consists of two steps: writing a printer-specific *render.c* function, and replacing the printer-specific values in *printertag.asm*. Note that a printer that does *not* support graphics has a very simple form of **Render()**; it returns an error. Here is sample code for **Render()** for a non-graphics printer (typically, an alphacom or diablo\_630):

```
#include "exec/types.h"
#include "devices/printer.h"
int
Render()
{
 return(PDERR_NOTGRAPHICS);
}
```

The following section describes the contents of a typical driver for a printer that actually supports graphics. The example code for the Epson printer, contained in the "Printer Device Source Code" appendix, shows a typical **Render()** function based on this description.

## Render.c

This function is the main printer-specific code module and consists of six parts:

- o Master initialization
- o Pixel rendering
- o Dumping a pixel buffer to the printer
- o Clearing and initializing the pixel buffer
- o Closing down
- o Density selection

**Master Initialization (case 0).** When this call is made, you are passed the width (in pixels) in *x* and the height (in pixels) in *y* of the picture as it should appear on the printer. Note that the printer non-specific code (using the printer-specific values in *printertag.asm* (that will be discussed later), has already verified that these values are within range for the printer. It is recommended that you use these values to allocate enough memory for a temporary buffer in which to build a command buffer for the printer. The buffer size needed is dependent on the specific printer, the width (usually), and the height (sometimes). In general, the buffer represents the commands and data required for one pass of the print head and usually takes the following form:

```
<start gfx cmd> <data> <end gfx cmd>
```

where:

```
<start
```

is the command required to define the graphic dump for each line.

```
<data>
```

is the binary data.

```
<end
```

is a terminator telling the printer to print the data (usually a carriage return).

For color printers, enough buffer space must usually be allocated for each different color ribbon, ink, and so on that the printer offers (the *okimate-20* and *diablo\_c-150* are provided as examples of this). Please refer to the sample drivers.

The example *render.c* functions use double buffering to reduce the dump time, which is why the `AllocMem()` call is for

(BUFSIZE times two)

where BUFSIZE represents the amount of memory for one entire print cycle (usually one pass of the print head).

Printers that would do more than one pass of the print head on a dump call are those that have to do a pass for each different main color that they want to lay down on the paper (like the Okidata-20 with three colors and the Epson\_jx-80 with four colors). A printer such as the Diablo\_c-150 that can lay down all the colors in a single pass needs to do only one pass.

The number of passes the printer has to do is irrelevant to you. This topic was introduced mainly to illustrate the true meaning of the term “one print cycle.” You want to send the printer an entire print cycle to allow the main non-printer-specific driver to continue onward, computing the values for the next print cycle while the printer is printing the previous dots. This is why you will find double buffering used in the example driver code.

Any other initialization that the printer requires should also be done at this time. It is advisable that you also do a reset command so that you know what state the printer is in before you try to send it any further commands.

In addition, after performing a reset command it is advisable to send no other commands for at least one second to allow the printer to “calm down”. Waiting after a reset is *strongly* recommended. The function `PWait(seconds,microseconds)` has been provided in the *wait.asm* file (see the “Printer Device Source Code” appendix) for this purpose. The *wait.asm* file must be assembled and linked into your custom printer device code.

**Render Pixel (Case 1).** When this call is made, your routine will be passed the x,y position of a single pixel and its color type. Note that the x,y value is an absolute value and you will have to do some modulus math (usually an AND) to compute the relative pixel position in your buffer. The absolute values will range from 0 to width-1 for x and 0 to height-1 for y. The color types are 0-black, 1-yellow, 2-magenta, and 3-cyan. Currently there is no provision for an RGB (red-green-blue) printer.

**Dump Buffer to Printer (Case 2).** When this call is made, you must send the buffer to the printer. As it now stands, there should be no need for you to change this routine. It should be common to all printers. It simply sends the buffer that you have been filling (via case 1) to the printer.

You would want to change this routine only if you need to do some post-processing on the buffer before it is sent to the printer. For example, if your printer uses the hexadecimal number \$03 as a command and requires that you send \$03 \$03 to send \$03 as data, you would probably want to scan the buffer and expand \$03's to \$03 \$03. Of course, you'll need to allocate space

somewhere in order to expand the buffer.

Because the printer driver does not send you the blank pixels, you must initialize the buffer to values for blank pixels (usually 0). Clearing the buffer should be the same for all printers. Initializing the buffer is printer-specific, and it includes placing the printer-specific control codes in the buffer ahead of and behind where the data will go.

**Closing Down (Case 3).** When this call is made you must wait for the print buffers to clear and then de-allocate the memory. This routine should be common to all printers. It simply waits for both buffers to empty, and then deallocates the memory that they used. There should be no need for you to change this routine. If you do change it, however, make sure that the amount of memory allocated for case 0 is deallocated by this routine.

**Pre-Master Initialization (Case 4).** Currently this option is implemented only on the HPLaserJet and HPLaserJet PLUS printers, although the call is made to each printer-specific driver. Ignoring it causes no problems as the call is made simply to give you a chance to select a different density from the default one. You should note that this call is made *before* the master initialization call (case 0) and gives you a chance to alter any variables that the master initialization may use to program the printer. Refer to the HPLaserJet PLUS printer driver for an example of density selection.

## **Printertag.asm**

The printer-specific values that need to be filled in here are as follows:

### **MaxXDots**

the maximum number of dots the printer can print across the page.

### **MaxYDots**

the maximum number of dots the printer can print down the page. Generally, if the printer supports roll or form feed paper, this value should be 0 indicating that there is no limit. If the printer has a definite y dots maximum (as the HPLaserJet does), this number should be entered here.

### **XDotsInch**

the dot density in x (for example, 120 dpi).

### **YDotsInch**

the dot density in y (for example, 144 dpi).

### **PrinterClass**

the printer class the printer falls into. Current choices are:

**PPC\_BWALPHA** - alphanumeric, no graphics.  
**PPC\_BWGFY** - black&white (only) graphics.  
**PPC\_COLORGFY** - color (and maybe b/w) graphics.

### **ColorClass**

the color class the printer falls into. Current choices are:

**PCC\_BW** - Black&White only (for example, EPSON).  
**PCC\_YMC** - Yellow Magenta Cyan only.  
**PCC\_YMC\_BW** - Yellow or Black&White but not both  
(for example, Okimate 20).  
**PCC\_YMCB** - YellowMagentaCyanBlack (for example, Diablo\_c-150).

### **NumRows**

the number of pixel rows printed by one pass of the print head. This number is used by the non-printer-specific code to determine when to make a case 2 (see above) call to you. You have to keep this number in mind when determining how big a buffer you'll need to store one print cycle's worth of data.

## **WRITING AN ALPHANUMERIC PRINTER DRIVER**

This alphanumeric section is meant to be read with the alpha listing for the EpsonX80 and Diablo Adv 25 close at hand.

The alphanumeric portion of the printer driver is designed to convert ANSI x3.64 style commands into the specific escape codes required by each individual printer. For example, the ANSI code for italics on is ESC[3m. The Epson FX80 printer would like a ESC%G to begin italic output mode. By using the printer driver all printers may be handled in a similar manner.

There are two parts to the alphanumeric portion of the printer driver: the **CommandTable** data table and the **DoSpecial()** routine.

### **Command Table**

The **CommandTable** is used to convert all escape codes that can be handled by simple substitution. It has one entry per ANSI command supported by the printer driver. When you are creating a custom **CommandTable**, you must maintain the order of the commands in the same sequence as that shown in *printer.h* and *printer.i*. By placing the specific codes for your printer in the proper position, the conversion takes place automatically.

*Note:* If the code for your printer requires a decimal 0 (an ASCII NULL character), you enter this NULL into the **CommandTable** as octal 376 (decimal 254).

Placing an octal value of 377 (255 decimal) in a position in the command table indicates to the printer device that no simple conversion is available on this printer for this ANSI command. For example, if a printer does not support one of the functions (for instance, if a daisy-wheel printer does not have a foreign character set), 377 octal (255 decimal) is placed in that position. However, 377 in a position can also mean that the ANSI function is to be handled by code located in the **DoSpecial()** function.

## **DoSpecial() Function**

The **DoSpecial()** function is meant to implement all the ANSI functions that cannot be done by simple substitution, but can be handled by a more complex sequence of control characters sent to the printer. These are functions that need parameter conversion, read values from Preferences, and so on.

The **DoSpecial()** function is set up as follows:

```
#include "exec/types.h"
#include "devices/printer.h"
#include "devices/prtbase.h"

extern struct PrinterData *PD;

DoSpecial(command,outputBuffer,vline,currentVMI,crlfFlag,Parms)
 char outputBuffer[];
 UWORD *command;
 BYTE *vline;
 BYTE *currentVMI;
 BYTE *crlfFlag;
 UBYTE Parms[];
{
 /* code begins here... */
```

where

### **command**

points to the command number. The *devices/printer.h* file contains the definitions for the routines to use (aRIN is initialize, and so on).

### **vline**

points to the value for the current line position.

**currentVMI**

points to the value for the current line spacing.

**crlfFlag**

points to the setting of the “add line feed after carriage return” flag.

**Parms**

contain whatever parameters were given with the ANSI command.

**outputBuffer**

points to the memory buffer into which the converted command is returned.

Almost every printer will require an aRIN (initialize) command in **DoSpecial()**. This command reads the printer settings from Preferences and creates the proper control sequence for the specific printer. Also, it returns the character set to normal (not italicized, not bold, and so on). Other functions depend on the printer.

Certain functions are implemented both in the **CommandTable** and in the **DoSpecial()** routine. These are functions such as superscript, subscript, PLU (partial line up), and PLD (partial line down), which can often be handled by a simple conversion. However, certain of these functions must also adjust the printer device’s line-position variable.





## **Chapter 16**

# **Clipboard Device**

### **Introduction**

The clipboard device is implemented as an Exec-style device. It is responsible for caching data that has been “cut” and providing data to “paste” in an application.

## Clipboard Commands

The clipboard responds to the following system functions:

**OpenDevice()**    Open the clipboard device  
**CloseDevice()**    Close the clipboard device  
**BeginIO()**        Initiate clipboard I/O  
**SendIO()**         Initiate a command and return immediately  
**DoIO()**

The I/O commands and their implementations are as follows:

**CMD\_INVALID**        Always an invalid command.

**CMD\_READ**            Read data from the clipboard for a paste. **io\_Offset** and **io\_ClipID** must be set to zero for the first read of a paste sequence. An **io\_Actual** that is less than the **io\_Length** indicates that all the data has been read. After all the data has been read, a subsequent read must be performed (one whose **io\_Actual** returns zero) to indicate to the clipboard device that all the data has been read. This allows random access of the clip while reading (provided only valid reads are performed).

**CMD\_WRITE**           Write data to the clipboard as a cut. **io\_Offset** and **io\_ClipID** must be set to zero for the first write of a cut sequence. An update command indicates that all the data has been written.

**CMD\_UPDATE**         Indicate that the data provided with a write command is complete and available for subsequent read/pastes.

**CMD\_CLEAR**           Clear any cut from this unit. Subsequent read/pastes will have no data available.

**CMD\_STOP**            Service no commands except invalid, start, flush.

**CMD\_START**           Resume command servicing.

**CMD\_FLUSH**           Abort all pending commands.

**CBD\_POST**            Post the availability of clip data. **io\_ClipID** must be set to zero, a subsequent write of this data does not have **io\_ClipID** set to zero as described above, but to the value in **io\_ClipID**.

**CMD\_CLIPREADID**     Return the **io\_ClipID** of the current clip to read.

**CMD\_CLIPWRITEID**    Return the **io\_ClipID** of the latest clip written.

## Clipboard Data

Data on the clipboard resides in one of three places. When an application posts a cut, the data resides in that application's private memory space. When an application writes to the clipboard, either of its own volition or in response to a message from the clipboard to satisfy a post, the data is copied to the clipboard, either to memory or to a special disk file. When the clipboard is not open, the data resides in the special disk file.

Data on the clipboard is self-identifying. It must be a correct IFF (Interchange Format Files) file; the rest of this section refers to IFF concepts. See the appendixes of the *Amiga ROM Kernel Reference Manual: Exec* for a complete description of IFF. If the top-level chunk is of type CAT or LIST with an identifier of CLIP, that indicates that the contained chunks are different representations of the same data, in decreasing order of preference on the part of the producer of the clip. Any other data is as defined elsewhere (probably a single representation of the cut data produced by an application).

The clipboard tool, which is the application that allows a Workbench user to view a clip, understands only the text (FTXT) and graphics (ILBM) form types. Applications using the clipboard to export data should include at least one of these types in a CLIP CAT so that their data can be represented on the clipboard in some form for user feedback.

The clipboard device nonstandard I/O request is called an **IOClipReq** and looks like a standard request except for the addition of the **io\_ClipID** field, which is assigned by the device to identify clips. It must be set to zero by the application for a post or an initial write or read, but preserved for subsequent writes or reads. The same initialization must be performed for the **io\_Offset** field, but for different reasons.

```
struct IOClipReq {
 struct Message io_Message;
 struct Device *io_Device; /* device node pointer */
 struct Unit *io_Unit; /* unit (driver private)*/
 UWORD io_Command; /* device command */
 UBYTE io_Flags; /* including QUICK and SATISFY */
 BYTE io_Error; /* error or warning num */
 ULONG io_Actual; /* number of bytes transferred */
 ULONG io_Length; /* number of bytes requested */
 SPTR io_Data; /* either clip stream or post port */
 ULONG io_Offset; /* offset in clip stream */
 LONG io_ClipID; /* ordinal clip identifier */
}
```

## Clipboard Messages

When an application performs a post, it must specify a message port for the clipboard to send a message to if it needs the application to satisfy the post with a write called the **SatisfyMsg**.

```
struct SatisfyMsg {
 struct Message sm_Message; /* the length will be 6 */
 UWORD sm_Unit; /* 0 for the primary clip unit */
 LONG sm_ClipID; /* the clip identifier of the post */
}
```

If the application wishes to determine if a post it has recently performed is still the current clip, it should check the **io\_ClipID** found in the post request upon return with that returned by the **CLIPREADID** command.

If an application has a pending post and wishes to determine if it should satisfy it (for example, before it exits), it should check the **io\_ClipID** of the post I/O request with that of the **CLIPWRITEID** command. If the application receives a satisfy message from the clipboard device (format described below), it must immediately perform the write with the **io\_ClipID** of the post. The satisfy message from the clipboard may be removed from the application message port by the clipboard device at any time (because it is re-used by the clipboard device). It is not dangerous to spuriously satisfy a post, however, because it is identified by the **io\_ClipID**.

The cut data is provided to the clipboard device via either a write or a post of the cut data. The write command accepts the data immediately and copies it onto the clipboard. The post command allows an application to inform the clipboard of a cut, but defers the write until the data is actually required for a paste. In the preceding discussion, references to the read and write commands of the clipboard device actually refer to a sequence of read or write commands, where the clip data is acquired and provided in pieces instead of all at once. The clipboard has an end-of-clip concept that is somewhat analogous to end-of-file for both read and write. The read end-of-file must be triggered by the user of the clipboard in order for the clipboard to move on to service other users' requests, and consists of reading data past the end of file. The write end-of-file is indicated by use of the update command, which indicates to the clipboard that the previous write commands are completed. See the description of the commands above for more information.

## Multiple Clips

The clipboard also supports multiple clips. This is not to be confused with the multiple IFF CLIP chunks in a clip, which allow for different representation of the same data. Multiple clips store different data. Applications performing cut and paste operations generally specify the primary clip. The alternate clips are provided to aid applications in the maintenance of a set of

clips (like a scrapbook). The multiple clips are implemented as different units in the clipboard device, and are thus accessed at open time:

```
OpenDevice("clipboard.device", unit, &IOClipReq, 0);
```

The primary clip unit used by applications to share data is unit 0; use of alternate clip units is by private convention.

## Example Program

```
#include "exec/types.h"
#include "graphics/gfx.h"
#include "graphics/gfxbase.h"
#include "graphics/view.h"
#include "intuition/intuition.h"
#include "libraries/dos.h"
#include "libraries/dosextens.h"
#include "devices/clipboard.h"

extern int stdout;
struct GfxBase *GfxBase;

char buffer[80], *b, c;
int rawConsole, oldStdout, postID;

readS()
{
 b = buffer;
 while (Read(rawConsole, &c, 1), ((c != '\34') && (c != '\r'))) {
 *b++ = c;
 printf("%lc", c);
 }
 *b = '\0';
}

main()
{
 int i;
 GfxBase = (struct GfxBase *) OpenLibrary("graphics.library", 0);

 printf("CBOpen returned %ld.\n", CBOpen(PRIMARY_CLIP));
}
```

```

printf(" CBOpen RAW: file is %lx.\n", rawConsole ==
 Open("RAW:25/25/615/150/clipboard.device test", MODE_OLDFILE));

oldStdout = stdout;
stdout = rawConsole;
printf("\033[20h");

c = 0;
postID = 0;
while (c != '\34') {
 while((postID) && (!WaitForChar(rawConsole, 1000000)))
 if (CBCheckSatisfy(&postID)) {
 if (postID) {
 printf("Satisfy post data\n");
 readS();
 printf("\nsatisfying \"%s\"\n", buffer);
 CBSatisfyPost(buffer);
 postID = 0;
 }
 }
 Read(rawConsole, &c, 1);
 switch (c) {
 case 'w':
 printf("Enter cut data\n");
 readS();
 printf("\ncutting \"%s\"\n", buffer);
 CBCutS(buffer);
 break;
 case 'r':
 CBPasteS(buffer);
 printf("paste is \"%s\"\n");
 break;
 case 'p':
 printf("Posting post...\n");
 postID = CBPost();
 break;
 default:;
 }
}

CBClose();
printf("CBClose returned.\n");

Close(rawConsole);
stdout = oldStdout;

```

```

 printf("\nTest Done.\n");
}

```

```

strcpy(to, from)
register char *to, *from;
{
 do {
 *to++ = *from;
 } while(*from++);
}

```

```

strcat(to, from)
register char *to, *from;
{
 while(*to) to++;

 strcpy(to, from);
}

```

```

strlen(s)
register char *s;
{
 register i = 0;

 while(*s++) i++;

 return(i);
}

```

```

strcmp(a, b)
register char *a, *b;
{
 while(*a++ == *b) {
 if(!*b++) return(0);
 }

 if(*--a < *b) return(-1);
 return(1);
}

```

```

char *
index(s, c)
char *s, c;
{
 char sc;

```



```

while(sc == *s) {
 if(sc == c) return(s);
 s++;
}
return(0);
}

```

```

char *
rindex(origs, c)
char *origs, c;
{
 char sc, *s;

 s = &origs[strlen(origs) - 1];

 while(s >= origs) {
 if(*s == c) return(s);
 s--;
 }
 return(0);
}

```

```

char *
TailPath(path)
char *path;
{
 char *last;

 /* looking for "volume:/name/bar/tail".
 * The routine breaks if volume has a slash...
 */

 /* check for a slash */
 if(!(last = rindex(path, '/'))) {

 /* no slash. Check for a colon */
 if(!(last = rindex(path, ':'))) {

 /* no colon either. Return the original */
 return(path);
 }
 }
 return(last);
}

```

## Support Functions Called from Example Program

```
/*
 * Program name: cbio
 * Purpose: Provide standard clipboard device interface routines
 * such as Open, Post, Read, Write, etc.
 */
#include "exec/types.h"
#include "exec/ports.h"
#include "exec/io.h"
#include "devices/clipboard.h"

struct IOClipReq clipboardIO = 0;
struct MsgPort clipboardMsgPort = 0;
struct MsgPort satisfyMsgPort = 0;

int CBOpen(unit)
int unit;
{
 int error;

 /* open the clipboard device */
 if ((error = OpenDevice("clipboard.device", unit, &clipboardIO, 0)) != 0)
 return(error);

 /* Set up the message port in the I/O request */
 clipboardMsgPort.mp_Node.ln_Type = NT_MSGPORT;
 clipboardMsgPort.mp_Flags = 0;
 clipboardMsgPort.mp_SigBit = AllocSignal(-1);
 clipboardMsgPort.mp_SigTask = (struct Task *) FindTask((char *) NULL);
 AddPort(&clipboardMsgPort);
 clipboardIO.io_Message.mn_ReplyPort = &clipboardMsgPort;

 satisfyMsgPort.mp_Node.ln_Type = NT_MSGPORT;
 satisfyMsgPort.mp_Flags = 0;
 satisfyMsgPort.mp_SigBit = AllocSignal(-1);
 satisfyMsgPort.mp_SigTask = (struct Task *) FindTask((char *) NULL);
 AddPort(&satisfyMsgPort);

 return(0);
}

CBClose()
{
 RemPort(&satisfyMsgPort);
 RemPort(&clipboardMsgPort);
}
```

```

 CloseDevice(&clipboardIO);
}

CBCut(stream, length)
char *stream;
int length;
{
 clipboardIO.io_Command = CMD_WRITE;
 clipboardIO.io_Data = stream;
 clipboardIO.io_Length = length;
 clipboardIO.io_Offset = 0;
 clipboardIO.io_ClipID = 0;
 DoIO(&clipboardIO);
 clipboardIO.io_Command = CMD_UPDATE;
 DoIO(&clipboardIO);
}

writeLong(ldata)
LONG *ldata;
{
 clipboardIO.io_Command = CMD_WRITE;
 clipboardIO.io_Data = ldata;
 clipboardIO.io_Length = 4;
 DoIO(&clipboardIO);
}

CBSatisfyPost(string)
char *string;
{
 int length;
 char *s;

 length = 0;
 s = string;
 while(*s++) length++;

 clipboardIO.io_Offset = 0;
 writeLong("FORM"); /* "FORM" */
 length += 12;
 writeLong(&length); /* # */
 writeLong("TEST"); /* "TEST" */
 writeLong("TEST"); /* "TEST" */
 length -= 12;
 writeLong(&length); /* # */

 clipboardIO.io_Command = CMD_WRITE;
 clipboardIO.io_Data = string;
}

```

```

 clipboardIO.io_Length = length;
 DoIO(&clipboardIO); /* text string */

 clipboardIO.io_Command = CMD_UPDATE;
 DoIO(&clipboardIO);
}

CBCutS(string)
char *string;
{
 clipboardIO.io_ClipID = 0;
 CBSatisfyPost(string);
}

CBPasteS(string)
char *string;
{
 int length;

 clipboardIO.io_Command = CMD_READ;
 clipboardIO.io_Data = 0;
 clipboardIO.io_Length = 16;
 clipboardIO.io_Offset = 0;
 clipboardIO.io_ClipID = 0;
 DoIO(&clipboardIO);

 clipboardIO.io_Command = CMD_READ;
 clipboardIO.io_Data = &length;
 clipboardIO.io_Length = 4;
 DoIO(&clipboardIO);

 clipboardIO.io_Command = CMD_READ;
 clipboardIO.io_Data = string;
 clipboardIO.io_Length = length;
 DoIO(&clipboardIO);

 string[length] = '\0';

 /* force end of file to terminate read */
 clipboardIO.io_Command = CMD_READ;
 clipboardIO.io_Length = 1;
 clipboardIO.io_Data = 0;
 DoIO(&clipboardIO);
}

int
CBPost()
{

```

```

clipboardIO.io_Command = CBD_POST;
clipboardIO.io_Data = &satisfyMsgPort;
clipboardIO.io_ClipID = 0;
DoIO(&clipboardIO);
return(clipboardIO.io_ClipID);
}

int
CBCurrentReadID()
{
 clipboardIO.io_Command = CMD_CLIPREADID;
 DoIO(&clipboardIO);
 return(clipboardIO.io_ClipID);
}

int
CBCurrentWriteID()
{
 clipboardIO.io_Command = CMD_CLIPWRITEID;
 DoIO(&clipboardIO);
 return(clipboardIO.io_ClipID);
}

BOOL
CBCheckSatisfy(idVar)
int *idVar;
{
 struct SatisfyMsg *sm;

 if (*idVar == 0)
 return(TRUE);
 if (*idVar < CBCurrentWriteID()) {
 *idVar = 0;
 return(TRUE);
 }
 if (sm = (struct SatisfyMsg *) GetMsg(&satisfyMsgPort)) {
 if (*idVar == sm->sm_ClipID)
 return(TRUE);
 }
 return(FALSE);
}

```



## **PART III**

# Chapter 17

## Math Functions

This chapter describes the structure and calling sequences required to access the Motorola Fast Floating Point and IEEE Double Precision math libraries via the Amiga-supplied interfaces.

### Introduction

In its present state, the FFP library consists of three separate entities: the basic math library, the transcendental math library, and C and assembly-language interfaces to the basic math library plus FFP conversion functions. The IEEE Double Precision library presently consists of one entity: the basic math library.



# FFP Floating Point Data Format

FFP floating-point variables are defined within C by the float or FLOAT directive. In assembly language they are simply defined by a DC.L/DS.L statement. All FFP floating-point variables are defined as 32-bit entities (longwords) with the following format:



where

M = 24-bit mantissa

S = Sign of FFP number

E = Exponent in excess-64 notation

The mantissa is considered to be a binary fixed-point fraction; except for 0, it is always normalized (has a 1 bit in its highest position). Thus, it represents a value of less than 1 but greater than or equal to 1/2.

The sign bit is reset (0) for a positive value and set (1) for a negative value.

The exponent is the power of two needed to correctly position the mantissa to reflect the number's true arithmetic value. It is held in excess-64 notation, which means that the two's-complement values are adjusted upward by 64, thus changing \$40 (-64) through \$3F (+63) to \$00 through \$7F. This facilitates comparisons among floating-point values.

The value of 0 is defined as all 32 bits being 0s. The sign, exponent, and mantissa are entirely cleared. Thus, 0s are always treated as positive.

The range allowed by this format is as follows:

DECIMAL:

$9.22337177 \times 10^{18} > +\text{VALUE} > 5.42101070 \times 10^{-20}$   
 $-9.22337177 \times 10^{18} < -\text{VALUE} < -2.71050535 \times 10^{-20}$

BINARY (HEXADECIMAL):

$.FFFFFF \times 2^{63} > +\text{VALUE} > .800000 \times 2^{-63}$   
 $-.FFFFFF \times 2^{63} < -\text{VALUE} < -.800000 \times 2^{-64}$

Remember that you cannot perform *any* arithmetic on these variables without using the fast floating-point libraries. The formats of the variables are *incompatible* with the arithmetic format of C-generated code; hence, all floating-point operations are performed through function calls.

## FFP Basic Mathematics Library

The FFP basic math library resides in ROM and is opened by making a call to the `OpenLibrary()` function with `mathffp.library` as the argument. In C, this might be implemented as shown below:

```
int MathBase;

main()
{
 char lib_name[] = "mathffp.library";

 if ((MathBase = OpenLibrary(lib_name, 0)) < 1) {
 printf("Can't open %s: vector = %08x\n", lib_name,
 MathBase);
 exit(); }
 .
 .
 .
}
```

The global variable **MathBase** is used internally for all future library references.

This library contains entries for the basic mathematics functions such as add, subtract, and so on. The C-called entry points are accessed via code generated by the C compiler when standard numerical operators are given within the source code. Note that to use either the C or assembly language interfaces to the basic math library all user code must be linked with the library *amiga.lib*. The C entry points defined for the basic math functions are as follows:

|                    |                                                                          |
|--------------------|--------------------------------------------------------------------------|
| <code>fixi</code>  | Convert FFP variable to integer<br>Usage: <code>i1 = (int) f1;</code>    |
| <code>flti</code>  | Convert integer variable to FFP<br>Usage: <code>f1 = (FLOAT) i1;</code>  |
| <code>fcmpi</code> | Compare two FFP variables<br>Usage: <code>if (f1 &lt;&gt; f2) {};</code> |
| <code>ftsti</code> | Test an FFP variable against zero<br>Usage: <code>if (!f1) {};</code>    |
| <code>fabsi</code> | Take absolute value of FFP variable<br>Usage: <code>f1 = abs(f2);</code> |
| <code>fnegi</code> | Take two's complement of FFP variable<br>Usage: <code>f1 = -f2;</code>   |
| <code>faddi</code> | Add two FFP variables<br>Usage: <code>f1 = f2 + f3;</code>               |
| <code>fsubi</code> | Subtract two FFP variables<br>Usage: <code>f1 = f2 - f3;</code>          |
| <code>fmul</code>  | Multiply two FFP variables<br>Usage: <code>f1 = f2 * f3;</code>          |
| <code>fdivi</code> | Divide two FFP variables<br>Usage: <code>f1 = f2 / f3;</code>            |

Be sure to include proper data type definitions as shown in the example below.

```

#include <libraries/mathffp.h>
int MathBase;

main()
{
 FLOAT f1, f2, f3;
 int i1, i2, i3;
 char lib_name[] = "mathffp.library";

 if((MathBase = OpenLibrary(lib_name, 0)) < 1) {
 printf(" Can't open %s: vector = %08x\n", lib_name,
 MathBase);
 exit(); }

 i1 = (int) f1; /* Call ffixi entry */
 fi = (FLOAT) i1; /* Call fflti entry */

 if (f1 < f2) {}; /* Call fcmpi entry */
 if (!f1) {}; /* Call ftsti entry */

 f1 = abs(f2); /* Call fabsi entry */
 f1 = -f2; /* Call fnegi entry */
 f1 = f2 + f3; /* Call faddi entry */
 f1 = f2 - f3; /* Call fsubi entry */
 f1 = f2 * f3; /* Call fmuli entry */
 f1 = f2 / f3; /* Call fdivi entry */
}

```

The Amiga assembly language interface to the Motorola Fast Floating Point basic math routines is shown below, including some details about how the system flags are affected by each operation. This interface resides in *amiga.lib* and must be linked with the user code. Note that the access mechanism from assembly language is as follows:

```

MOVEA.L _MathBase,A6
JSR _LVOSPFix,A6

```

\_LVOSPFix - Convert FFP to integer

Inputs: D0 = FFP argument  
Outputs: D0 = Integer (two's complement) result  
Condition codes: N = 1 if result is negative  
Z = 1 if result is zero  
V = 1 if overflow occurred  
C = undefined  
X = undefined

\_LVOSPFIt - Convert integer to FFP

Inputs: D0 = Integer (two's complement) argument  
Outputs: D0 = FFP result  
Condition codes: N = 1 if result is negative  
Z = 1 if result is zero  
V = 0  
C = undefined  
X = undefined

\_LVOSPCmp - Compare

Inputs: D1 = FFP argument 1  
D0 = FFP argument 2  
Outputs: D0 = +1 if  $\text{arg1} < \text{arg2}$   
D0 = -1 if  $\text{arg1} > \text{arg2}$   
D0 = 0 if  $\text{arg1} = \text{arg2}$   
Condition codes: N = 0  
Z = 1 if result is zero  
V = 0  
C = undefined  
X = undefined  
GT =  $\text{arg2} > \text{arg1}$   
GE =  $\text{arg2} \geq \text{arg1}$   
EQ =  $\text{arg2} = \text{arg1}$   
NE =  $\text{arg2} <> \text{arg1}$   
LT =  $\text{arg2} < \text{arg1}$   
LE =  $\text{arg2} \leq \text{arg1}$

\_LVOSPst - Test

Inputs: D1 = FFP argument  
Outputs: D0 = +1 if arg > 0.0  
D0 = -1 if arg < 0.0  
D0 = 0 if arg = 0.0  
Condition codes: N = 1 if result is negative  
Z = 1 if result is zero  
V = 0  
C = undefined  
X = undefined  
EQ = arg = 0.0  
NE = arg <> 0.0  
PL = arg >= 0.0  
MI = arg < 0.0

*Note:* This routine trashes the argument in D1.

\_LVOSPAbs - Absolute value

Inputs: D0 = FFP argument  
Outputs: D0 = FFP absolute value result  
Condition codes: N = 0  
Z = 1 if result is zero  
V = 0  
C = undefined  
X = undefined

\_LVOSPNeg - Negate

Inputs: D0 = FFP argument  
Outputs: D0 = FFP negated result  
Condition codes: N = 1 if result is negative  
Z = 1 if result is zero  
V = 0  
C = undefined  
X = undefined

**\_LVOSPPAdd - Addition**

Inputs: D1 = FFP argument 1  
D0 = FFP argument 2

Outputs: D0 = FFP addition of arg1+arg2 result

Condition codes: N = 1 if result is negative  
Z = 1 if result is zero  
V = 1 if result overflowed  
C = undefined  
Z = undefined

**\_LVOSPPSub - Subtraction**

Inputs: D1 = FFP argument 1  
D0 = FFP argument 2

Outputs: D0 = FFP subtraction of arg2-arg1 result

Condition codes: N = 1 if result is negative  
Z = 1 if result is zero  
V = 1 if result overflowed  
C = undefined  
Z = undefined

**\_LVOSPPMul - Multiply**

Inputs: D0 = FFP argument 1  
D2 = FFP argument 2

Outputs: D0 = FFP multiplication of arg1\*arg2 result

Condition codes: N = 1 if result is negative  
Z = 1 if result is zero  
V = 1 if result overflowed  
C = undefined  
Z = undefined

**\_LVOSPPDiv - Divide**

Inputs: D1 = FFP argument 1  
D0 = FFP argument 2

Outputs: D0 = FFP division of arg2/arg1 result

Condition codes: N = 1 if result is negative  
Z = 1 if result is zero  
V = 1 if result overflowed  
C = undefined  
Z = undefined

# FFP Transcendental Mathematics Library

The FFP transcendental math library resides on disk and must be accessed in the same way as the basic math library after it is loaded into system RAM. The name to be included in the `OpenLibrary()` call is *mathtrans.library*. In C, this might be implemented as follows:

```
int MathBase;
int MathTransBase;

main()
{
 char bmath_name[] = "mathffp.library";
 char tmath_name[] = "mathtrans.library";

 if((MathBase = OpenLibrary(bmath_name, 0)) < 1) {
 printf("Can't open %s: vector = %08x\n", bmath_name,
 MathBase);
 exit(); }

 if((MathTransBase = OpenLibrary(tmath_name, 0)) < 1) {
 printf("Can't open %s: vector = %08x\n", tmath_name,
 MathTransBase);
 exit(); }
 .
 .
 .
}
```

The global variables `MathBase` and `MathTransBase` are used internally for all future library references. Note that the transcendental math library is dependent upon the basic math library and, therefore, is opened after the basic math library has been opened.

This library contains entries for the transcendental math functions sine, cosine, and so on. The C-called entry points are accessed via code generated by the C compiler when the actual function names are given within the source code. The C entry points defined for the transcendental math functions are as follows:

- SPAsin     Return arcsine of FFP variable.  
          Usage: f1 = SPAsin(f2);
  
- SPAcos     Return arccosine of FFP variable.  
          Usage: f1 = SPAcos(f2);



- SPAtan Return arctangent of FFP variable.  
Usage:  $f1 = \text{SPAtan}(f2);$
- SPSin Return sine of FFP variable. This function accepts an FFP radian argument and returns the trigonometric sine value. For extremely large arguments where little or no precision would result, the computation is aborted and the "V" condition code is set. A direct return to the caller is made.  
Usage:  $f1 = \text{SPSin}(f2);$
- SPCos Return cosine of FFP variable. This function accepts an FFP radian argument and returns the trigonometric cosine value. For extremely large arguments where little or no precision would result, the computation is aborted and the "V" condition code is set. A direct return to the caller is made.  
Usage:  $f1 = \text{SPCos}(f2);$
- SPTan Return tangent of FFP variable. This function accepts an FFP radian argument and returns the trigonometric tangent value. For extremely large arguments where little or no precision would result, the computation is aborted and the "V" condition code is set. A direct return to the caller is made.  
Usage:  $f1 = \text{SPTan}(f2);$
- SPSincos Return sine and cosine of FFP variable. This function accepts an FFP radian argument and returns both the trigonometric sine and cosine values. If both the sine and cosine are required for a single radian value of interest, this function will result in almost twice the execution speed of calling the sin and cos functions independently. For extremely large arguments where little or no precision would result, the computation is aborted and the "V" condition code is set. A direct return to the caller is made.  
Usage:  $f1 = \text{SPSincos}(\&f3, f2);$
- SPSinh Return hyperbolic sine of FFP variable.  
Usage:  $f1 = \text{SPSinh}(f2);$
- SPCosh Return hyperbolic cosine of FFP variable.  
Usage:  $f1 = \text{SPCosh}(f2);$
- SPTanh Return hyperbolic tangent of FFP variable.  
Usage:  $f1 = \text{SPTanh}(f2);$
- SPExp Return  $e$  to the FFP variable power. This function accepts an FFP argument and returns the result representing the value of  $e$  (2.71828...) raised to that power.

Usage: `f1 = SPExp(f2);`

**SPLog** Return natural log (base *e*) of FFP variable.  
Usage: `f1 = SPLog(f2);`

**SPLog10** Return naparian log (base 10) of FFP variable.  
Usage: `f1 = SPLog10(f2);`

**SPPow** Return FFP arg2 to FFP arg1.  
Usage: `f1 = SPPow(f3, f2);`

**SPSqrt** Return square root of FFP variable.  
Usage: `f1 = SPSqrt(f2);`

**SPTieee** Convert FFP variable to IEEE format  
Usage: `i1 = SPTieee(f1);`

**SPFieee** Convert IEEE variable to FFP format.  
Usage: `f1 = SPFieee(i1);`

Be sure to include proper data type definitions, as shown in the example below.

```

#include <mathffp.h>

int MathBase;
int MathTransBase;

main()
{
 FLOAT f1, f2, f3;
 int i1, i2, i3;
 char bmath_name[] = "mathffp.library";
 char tmath_name[] = "mathtrans.library";

 if((MathBase = OpenLibrary(bmath_name, 0)) < 1) {
 printf("Can't open %s: vector = %08x\n", bmath_name, MathBase);
 exit(); }

 if((MathTransBase = OpenLibrary(tmath_name, 0)) < 1) {
 printf("Can't open %s: vector = %08x\n", tmath_name, MathTransBase);
 exit(); }

 f1 = SPAsin(f2); /* Call SPAsin entry */

```

```

f1 = SPACos(f2); /* Call SPACos entry */
f1 = SPAtan(f2); /* Call SPAtan entry */

f1 = SPSin(f2); /* Call SPSin entry */
f1 = SPCos(f2); /* Call SPCos entry */
f1 = SPTan(f2); /* Call SPTan entry */
f1 = SPSincos(&f3, f2); /* Call SPSincos entry */

f1 = SPSinh(f2); /* Call SPSinh entry */
f1 = SPCosh(f2); /* Call SPCosh entry */
f1 = SPTanh(f2); /* Call SPTanh entry */

f1 = SPExp(f2); /* Call SPExp entry */
f1 = SPLog(f2); /* Call SPLog entry */
f1 = SPLog10(f2); /* Call SPLog10 entry */
f1 = SPPow(f2); /* Call SPPow entry */
f1 = SPSqrt(f2); /* Call SPSqrt entry */

i1 = SPTieee(f2); /* Call SPTieee entry */
f1 = SPFieee(i1); /* Call SPFieee entry */
}

```

The section below describes the Amiga assembly language interface to the Motorola Fast Floating Point transcendental math routines and includes some details about how the system flags are affected by the operation. Again, this interface resides in the library file *mathlink.lib* and must be linked with the user code. Note that the access mechanism from assembly language is as shown below:

```

LEA _LVOSPAsin,A6
JSR _MathTransBase(A6)

```

LVOSPAsin - Arcsine

|                  |                                   |
|------------------|-----------------------------------|
| Inputs:          | D0 = FFP argument                 |
| Outputs:         | D0 = FFP arctangent radian result |
| Condition codes: | N = 0                             |
|                  | Z = 1 if result is zero           |
|                  | V = 0                             |
|                  | C = undefined                     |
|                  | X = undefined                     |

**\_LVOSPAcos - Arccosine**

|                  |                                   |
|------------------|-----------------------------------|
| Inputs:          | D0 = FFP argument                 |
| Outputs:         | D0 = FFP arctangent radian result |
| Condition codes: | N = 0                             |
|                  | Z = 1 if result is zero           |
|                  | V = 0                             |
|                  | C = undefined                     |
|                  | X = undefined                     |

**\_LVOSPAtan - Arctangent**

|                  |                                   |
|------------------|-----------------------------------|
| Inputs:          | D0 = FFP argument                 |
| Outputs:         | D0 = FFP arctangent radian result |
| Condition codes: | N = 0                             |
|                  | Z = 1 if result is zero           |
|                  | V = 0                             |
|                  | C = undefined                     |
|                  | X = undefined                     |

**\_LVOSPSin - Sine**

|                  |                                                                        |
|------------------|------------------------------------------------------------------------|
| Inputs:          | D0 = FFP argument in radians                                           |
| Outputs:         | D0 = FFP sine result                                                   |
| Condition codes: | N = 1 if result is negative                                            |
|                  | Z = 1 if result is zero                                                |
|                  | V = 1 if result is meaningless<br>(that is, input magnitude too large) |
|                  | C = undefined                                                          |
|                  | X = undefined                                                          |

**\_LVOSPCos - Cosine**

|                  |                                                                        |
|------------------|------------------------------------------------------------------------|
| Inputs:          | D0 = FFP argument in radian                                            |
| Outputs:         | D0 = FFP cosine result                                                 |
| Condition codes: | N = 1 if result is negative                                            |
|                  | Z = 1 if result is zero                                                |
|                  | V = 1 if result is meaningless<br>(that is, input magnitude too large) |
|                  | C = undefined                                                          |
|                  | X = undefined                                                          |

`_LVOSPtan` - Tangent

|                  |                                                                        |
|------------------|------------------------------------------------------------------------|
| Inputs:          | D0 = FFP argument in radians                                           |
| Outputs:         | D0 = FFP tangent result                                                |
| Condition codes: | N = 1 if result is negative                                            |
|                  | Z = 1 if result is zero                                                |
|                  | V = 1 if result is meaningless<br>(that is, input magnitude too large) |
|                  | C = undefined                                                          |
|                  | X = undefined                                                          |

`_LVOSPSincos` - Sine and cosine

|                  |                                                                        |
|------------------|------------------------------------------------------------------------|
| Inputs:          | D0 = FFP argument in radians                                           |
|                  | D1 = Address to store cosine result                                    |
| Outputs:         | D0 = FFP sine result                                                   |
|                  | (D1) = FFP cosine result                                               |
| Condition codes: | N = 1 if result is negative                                            |
|                  | Z = 1 if result is zero                                                |
|                  | V = 1 if result is meaningless<br>(that is, input magnitude too large) |
|                  | C = undefined                                                          |
|                  | X = undefined                                                          |

`_LVOSPSinh` - Hyperbolic sine

|                  |                                 |
|------------------|---------------------------------|
| Inputs:          | D0 = FFP argument in radians    |
| Outputs:         | D0 = FFP hyperbolic sine result |
| Condition codes: | N = 1 if result is negative     |
|                  | Z = 1 if result is zero         |
|                  | V = 1 if overflow occurred      |
|                  | C = undefined                   |
|                  | X = undefined                   |

`_LVOSPCosh` - Hyperbolic cosine

|                  |                                   |
|------------------|-----------------------------------|
| Inputs:          | D0 = FFP argument in radians      |
| Outputs:         | D0 = FFP hyperbolic cosine result |
| Condition codes: | N = 1 if result is negative       |
|                  | Z = 1 if result is zero           |
|                  | V = 1 if overflow occurred        |
|                  | C = undefined                     |
|                  | X = undefined                     |

`_LVOSP`Tanh - Hyperbolic tangent

|                  |                                    |
|------------------|------------------------------------|
| Inputs:          | D0 = FFP argument in radians       |
| Outputs:         | D0 = FFP hyperbolic tangent result |
| Condition codes: | N = 1 if result is negative        |
|                  | Z = 1 if result is zero            |
|                  | V = 1 if overflow occurred         |
|                  | C = undefined                      |
|                  | X = undefined                      |

`_LVOSPE`xp - Exponential

|                  |                             |
|------------------|-----------------------------|
| Inputs:          | D0 = FFP argument           |
| Outputs:         | D0 = FFP exponential result |
| Condition codes: | N = 0                       |
|                  | Z = 1 if result is zero     |
|                  | V = 1 if overflow occurred  |
|                  | C = undefined               |
|                  | Z = undefined               |

`_LVOSPL`og - Natural logarithm

|                  |                                    |
|------------------|------------------------------------|
| Inputs:          | D0 = FFP argument                  |
| Outputs:         | D0 = FFP natural logarithm result  |
| Condition codes: | N = 1 if result is negative        |
|                  | Z = 1 if result is zero            |
|                  | V = 1 if argument negative or zero |
|                  | C = undefined                      |
|                  | Z = undefined                      |

`_LVOSPL`og10 - Naparian (base 10) logarithm

|                  |                                    |
|------------------|------------------------------------|
| Inputs:          | D0 = FFP argument                  |
| Outputs:         | D0 = FFP natural logarithm result  |
| Condition codes: | N = 1 if result is negative        |
|                  | Z = 1 if result is zero            |
|                  | V = 1 if argument negative or zero |
|                  | C = undefined                      |
|                  | Z = undefined                      |

|                                                                                                              |                                                                                                                                                                                                                                                      |
|--------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>_LVOSPPow</b> - Power</p> <p>Inputs:</p> <p>Outputs:</p> <p>Condition codes:</p>                       | <p>D1 = FFP argument value</p> <p>D0 = FFP exponent value</p> <p>D0 = FFP result of arg taken to exp power</p> <p>N = 0</p> <p>Z = 1 if result is zero</p> <p>V = 1 if result overflowed or arg &lt; 0</p> <p>C = undefined</p> <p>Z = undefined</p> |
| <p><b>_LVOSPSqrt</b> - Square root</p> <p>Inputs:</p> <p>Outputs:</p> <p>Condition codes:</p>                | <p>D0 = FFP argument</p> <p>D0 = FFP square root result</p> <p>N = 0</p> <p>Z = 1 if result is zero</p> <p>V = 1 if argument was negative</p> <p>C = undefined</p> <p>Z = undefined</p>                                                              |
| <p><b>_LVOSP ieee</b> - Convert to IEEE format</p> <p>Inputs:</p> <p>Outputs:</p> <p>Condition codes:</p>    | <p>D0 = FFP format argument</p> <p>D0 = IEEE floating-point format result</p> <p>N = 1 if result is negative</p> <p>Z = 1 if result is zero</p> <p>V = undefined</p> <p>C = undefined</p> <p>Z = undefined</p>                                       |
| <p><b>_LVOSPF ieee</b> - Convert from IEEE format</p> <p>Inputs:</p> <p>Outputs:</p> <p>Condition codes:</p> | <p>D0 = IEEE floating-point format argument</p> <p>D0 = FFP format result</p> <p>N = undefined</p> <p>Z = 1 if result is zero</p> <p>V = 1 if result overflowed FFP format</p> <p>C = undefined</p> <p>Z = undefined</p>                             |

# FFP Mathematics Conversion Library

The FFP mathematics conversion library is accessed by linking code into the executable file being created. The name of the file to include in the library description of the link command line is *mathlink\_lib.lib*. When this is included, direct calls are made to the conversion functions. Only a C interface exists for the conversion functions; there is no assembly language interface. The basic math library is required in order to access these functions and might be opened as shown below.

```
int MathBase;

main()
{
 char bmath_name[] = "mathffp.library";

 if ((MathBase = OpenLibrary(bmath_name, 0)) < 1) {
 printf("Can't open %s: vector = %08x\n", bmath_name,
 MathBase);
 exit(); }

 .
 .
 .
}
```

The global variable **MathBase** is used internally for all future basic math library references.

This library contains entries for the conversion functions associated with math library usage. The C-called entry points are accessed via code generated by the C compiler when the actual function names are given within the source code. The C entry points defined for the math conversion functions are as follows:

- afp      Convert ASCII string into FFP equivalent.  
Usage: fnum = afp(&string[0]);
  
- fpa      Convert FFP variable into ASCII equivalent.  
Usage: exp = fpa(fnum, &string[0]);
  
- arnd     Round ASCII representation of FFP number.  
Usage: arnd(place, exp, &string[0]);



- dbf        Convert FFP dual-binary number to FFP equivalent.  
           Usage: fnum = dbf(exp, mant);
- fpbcd     Convert FFP variable to BCD equivalent.  
           Usage: fpbcd(fnum, &string[0]);

Be sure to include proper data type definitions, as shown in the example below. Print statements have been included to help clarify the format of the math conversion function calls.

```
#include <mathfp.h>

char st1[80] = "3.1415926535897";
char st2[80] = "2.718281828459045";
char st3[80], st4[80];

int MathBase;

main()
{
 FLOAT num1, num2, num3, num4, num5, num6, num7, num8, num9;
 FLOAT n1, n2, n3, n4, n5, n6, n7, n8, n9;
 int i1, i2, i3, i4, i5, i6, i7, i8, i9;
 int exp1, exp2, exp3, exp4, mant1, mant2,
 mant3, mant4, place1, place2;

 if ((MathBase=OpenLibrary("mathfp.library",0)) < 1) {
 printf("Can't open mathfp.library:vector =%08x\n",
 MathBase);
 exit();
 }

 n1 = afp(st1); /* Call afp entry */
 n2 = afp(st2); /* Call afp entry */
 printf("\n\nASCII %s converts to floating point %f",
 st1, n1);
 printf("\n\nASCII %s converts to floating point %f",
 st2, n2);

 num1 = 3.1415926535897;
 num2 = 2.718281828459045;

 exp1 = fpa(num1, st3); /* Call fpa entry */
 exp2 = fpa(num2, st4); /* Call fpa entry */
 printf("\n\nfloating point %f converts to ASCII %s", num1, st3);
```

```

printf("\nfloating point %f converts to ASCII %s",
 num2, st4);

place1 = -2;
place2 = -1;
arnd(place1, exp1, st3); /* Call arnd entry */
arnd(place2, exp2, st4); /* Call arnd entry */
printf("\nASCII round of %f to %d places yields %s",
 num1, place1, st3);
printf("\nASCII round of %f to %d places yields %s",
 num2, place2, st4);

exp1 = -3; exp2 = 3; exp3 = -3; exp4 = 3;
mant1 = 12345; mant2 = -54321; mant3 = -12345;
t4 = 54321;
n1 = dbf(exp1, mant1); /* Call dbf entry */
n2 = dbf(exp2, mant2); /* Call dbf entry */
n3 = dbf(exp3, mant3); /* Call dbf entry */
n4 = dbf(exp4, mant4); /* Call dbf entry */
printf("\n\ndbf of exp = %d and mant = %d yields FFP number
 of %f", exp1, mant1, n1);
printf("\n\ndbf of exp = %d and mant = %d yields FFP number
 of %f", exp2, mant2, n2);
printf("\n\ndbf of exp = %d and mant = %d yields FFP number
 of %f", exp3, mant3, n3);
printf("\n\ndbf of exp = %d and mant = %d yields FFP number
 of %f", exp4, mant4, n4);

num1 = -num1;
fpbcd(num1, st3); /* Call fpbcd entry */
st3[8] = '\0';
strcpy(&i2, &st3[4]);
st3[4] = '\0';
strcpy(&i1, st3);
printf("\n\nfloating point %f converts to BCD %08x%08x", num1, i1, i2);
num2 = -num2;
fpbcd(num2, st4); /* Call fpbcd entry */
st4[8] = '\0';
strcpy(&i4, &st4[4]);
st4[4] = '\0';
strcpy(&i3, st4);
printf("\n\nfloating point %f converts to BCD
 %08x%08x", num2, i3, i4);
}

```

# IEEE Double-precision Basic Math Library

The IEEE double-precision basic math library resides on disk and is opened by making a call to the `OpenLibrary()` function with `mathieeedoubbas.library` as the argument. In C, this might be implemented as shown below.

```
int MathIeeeDoubBasBase;

main()
{
 char lib_name[] = "mathieeedoubbas.library";

 if ((MathIeeeDoubBasBase = OpenLibrary(lib_name, 0)) < 1) {
 printf("Can't open %s: vector = %08x\n", lib_name,
 MathIeeeDoubBasBase);
 exit(); }
 .
 .
 .
}
```

The global variable `MathIeeeDoubBasBase` is used internally for all future library references.

This library contains entries for the basic mathematics functions, such as add, subtract, and so on. The C-called entry points are accessed via code generated by the C compiler when the actual function names are given within the source code. The C entry points defined for the IEEE double-precision basic math functions are listed below:

## IEEEDPFix

Convert IEEE double-precision variable to integer

Usage: `i1 = IEEEDPFix(f1);`

## IEEEDPFlt

Convert integer variable to IEEE double precision

Usage: `f1 = IEEEDPFlt(i1);`

## IEEEDPCmp

Compare two IEEE double-precision variables

Usage: `switch (IEEEDPCmp(f1, f2)) {};`

## IEEEDPtst

Test an IEEE double-precision variable against zero

Usage:    switch (IEEEEDPTst(f1)) {};

#### IEEEEDPAbs

Take absolute value of IEEE double-precision variable

Usage:    f1 = IEEEEDPAbs(f2);

#### IEEEEDPNeg

Take two's complement of IEEE double-precision variable

Usage:    f1 = IEEEEDPNeg(f2);

#### IEEEEDPAdd

Add two IEEE double-precision variables

Usage:    f1 = IEEEEDPAdd(f2, f3);

#### IEEEEDPSub

Subtract two IEEEEDPSub variables

Usage:    f1 = IEEEEDPSub(f2, f3);

#### IEEEEDPMul

Multiply two IEEE double-precision variables

Usage:    f1 = IEEEEDPMul(f2, f3);

#### IEEEEDPDiv

Divide two IEEE double-precision variables

Usage:    f1 = IEEEEDPDiv(f2, f3);

Be sure to include proper data type definitions, as shown in the example below.

```
int MathIeeeDoubBasBase;
```

```
main()
```

```
{
```

```
 double f1, f2, f3;
```

```
 int i1, i2, i3;
```

```
 char lib_name[] = "mathieeedoubbas.library";
```

```
 if((MathIeeeDoubBasBase = OpenLibrary(lib_name, 0)) < 1) {
```

```
 printf("Can't open %s: vector = %08x\n", lib_name,
```

```
 MathIeeeDoubBasBase);
```

```
 exit(); }
```

```
 i1 = IEEEEDPFix(f1);
```

```
 /* Call IEEEEDPFix entry */
```

```
 fi = IEEEEDPFIt(i1);
```

```
 /* Call IEEEEDPFIt entry */
```

```

switch (IEEEEDPCmp(f1, f2)) {}; /* Call IEEEEDPCmp entry */
switch (IEEEEDPTst(f1)) {}; /* Call IEEEEDPTst entry */
f1 = IEEEEDPAbs(f2); /* Call IEEEEDPAbs entry */
f1 = IEEEEDPNeg(f2); /* Call IEEEEDPNeg entry */
f1 = IEEEEDPAdd(f2, f3); /* Call IEEEEDPAdd entry */
f1 = IEEEEDPSub(f2, f3); /* Call IEEEEDPSub entry */
f1 = IEEEEDPMul(f2, f3); /* Call IEEEEDPMul entry */
f1 = IEEEEDPDiv(f2, f3); /* Call IEEEEDPDiv entry */
}

```

The Amiga assembly language interface to the IEEE double-precision floating-point basic math routines is shown below, including some details about how the system flags are affected by each operation. Note that the access mechanism from assembly language is as shown below:

```

LEA _LVOIEEEEDPFix,A6
JSR _MathIeeeDoubBasBase(A6)

```

\_LVOIEEEEDPFix - Convert IEEE double-precision to integer

```

Inputs: D0/D1 = IEEE double-precision argument
Outputs: D0 = Integer (two's complement) result
Condition codes: N = 1 if result is negative
 Z = 1 if result is zero
 V = 1 if overflow occurred
 C = undefined
 X = undefined

```

\_LVOIEEEEDPFIt - Convert integer to IEEE double-precision

```

Inputs: D0 = Integer (two's complement) argument
Outputs: D0/D1 = IEEE double-precision result
Condition codes: N = 1 if result is negative
 Z = 1 if result is zero
 V = 0
 C = undefined
 X = undefined

```

LVOIEEEDPCmp - Compare two IEEE double-precision values

Inputs: D0/D1 = IEEE double-precision argument 1  
D2/D3 = IEEE double-precision argument 2

Outputs: D0 = +1 if  $\text{arg1} < \text{arg2}$   
D0 = -1 if  $\text{arg1} > \text{arg2}$   
D0 = 0 if  $\text{arg1} = \text{arg2}$

Condition codes: N = 0  
Z = 1 if result is zero  
V = 0  
C = undefined  
X = undefined  
GT =  $\text{arg2} > \text{arg1}$   
GE =  $\text{arg2} \geq \text{arg1}$   
EQ =  $\text{arg2} = \text{arg1}$   
NE =  $\text{arg2} < > \text{arg1}$   
LT =  $\text{arg2} < \text{arg1}$   
LE =  $\text{arg2} \leq \text{arg1}$

LVOIEEEDPTst - Test an IEEE double-precision value against zero

Inputs: D0/D1 = IEEE double-precision argument

Outputs: D0 = +1 if  $\text{arg} > 0.0$   
D0 = -1 if  $\text{arg} < 0.0$   
D0 = 0 if  $\text{arg} = 0.0$

Condition codes: N = 1 if result is negative  
Z = 1 if result is zero  
V = 0  
C = undefined  
X = undefined  
EQ =  $\text{arg} = 0.0$   
NE =  $\text{arg} < > 0.0$   
PL =  $\text{arg} \geq 0.0$   
MI =  $\text{arg} < 0.0$

\_LVOIEEEDPAbs - Absolute value

Inputs: D0/D1 = IEEE double-precision argument  
Outputs: D0/D1 = IEEE double-precision absolute value result  
Condition codes: N = 0  
Z = 1 if result is zero  
V = 0  
C = undefined  
X = undefined

\_LVOIEEEDPNeg - Negate

Inputs: D0/D1 = IEEE double-precision argument  
Outputs: D0/D1 = IEEE double-precision negated result  
Condition codes: N = 1 if result is negative  
Z = 1 if result is zero  
V = 0  
C = undefined  
X = undefined

\_LVOIEEEDPAdd - Addition

Inputs: D0/D1 = IEEE double-precision argument 1  
D2/D3 = IEEE double-precision argument 2  
Outputs: D0/D1 = IEEE double-precision addition of arg1+arg2 result  
Condition codes: N = 1 if result is negative  
Z = 1 if result is zero  
V = 1 if result overflowed  
C = undefined  
Z = undefined

LVOIEEEDPSub - Subtraction

Inputs: D0/D1 = IEEE double-precision argument 1  
D2/D3 = IEEE double-precision argument 2  
Outputs: D0/D1 = IEEE double-precision subtraction  
of arg1-arg2 result  
Condition codes: N = 1 if result is negative  
Z = 1 if result is zero  
V = 1 if result overflowed  
C = undefined  
Z = undefined

LVOIEEEDPMul - Multiply

Inputs: D0/D1 = IEEE double-precision argument 1  
D2/D3 = IEEE double-precision argument 2  
Outputs: D0/D1 = IEEE double-precision multiplication  
of arg1\*arg2 result  
Condition codes: N = 1 if result is negative  
Z = 1 if result is zero  
V = 1 if result overflowed  
C = undefined  
Z = undefined

LVOIEEEDPDiv - Divide

Inputs: D0/D1 = IEEE double-precision argument 1  
D2/D3 = IEEE double-precision argument 2  
Outputs: D0/D1 = IEEE double-precision division  
of arg1/arg2 result  
Condition codes: N = 1 if result is negative  
Z = 1 if result is zero  
V = 1 if result overflowed  
C = undefined  
Z = undefined





# Chapter 18

## Workbench

This chapter shows how to use the Workbench facilities in your applications. For information about *IconEd*, the icon editor for making Workbench icons, see the appendixes of the *Introduction to Amiga* manual for revision 1.1 of the system software.

### Introduction

Workbench is both an application program and a screen in which other applications can run. Workbench allows users to interact with the Amiga file system by using icons, and it gives the programmer access to a body of library functions for manipulating the application's objects and icons.

Here are definitions of some terms that may be unfamiliar or used in unfamiliar ways in this chapter.

*Workbench object*

A Workbench object contains all the information that Workbench needs to display and use a project, tool, drawer, etc. The two kinds of Workbench objects are **WObject** (as Workbench uses objects) and **DiskObject** (as most other users will view objects in memory or in a file on disk).

*icon*

This is a shorthand name for a Workbench object. An icon may be in memory or on disk or both.

*info file*

The disk representation of an icon. The format of an icon on disk is slightly different from an icon in memory, but one is obtainable from the other.

*strings*

A null-terminated sequence of bytes.

*activating*

The act of starting a tool, opening a drawer, and so on. The term *opening* is reserved for windows and files.

*tool*

An application program or system utility.

*project*

Something produced by an executable program and associated with an executable program, for example, a text file or a drawing.

*drawer*

A disk-based directory.

## The Icon Library

The icon library, *icon.library*, has memory-management routines, icon input and output routines, and string manipulation routines. The "Library Summaries" appendix to this manual contains the reference pages for this library.

# The Info File

The *info file* is the center of interaction between applications and Workbench. This file stores all the necessary information to display an icon and to start up an application. An info file can contain several different types of icons, as shown in table 17-1.

Table 18-1: Contents of a Workbench Info File

| Icon Name | Object                      |
|-----------|-----------------------------|
| WBDISK    | The root of a disk          |
| WBDRAWER  | A directory on the disk     |
| WBTOOL    | A directly runnable program |
| WBPROJECT | A data file of some sort    |
| WBGARBAGE | The trash can directory     |
| WBKICK    | A non-DOS disk              |

The actual data present in the info file depends on the icon type. Note that *any* graphical image can be used for any icon type in the info file. In fact, the graphical image need not be unique for each type of icon. However, it is strongly recommended as a matter of programming style that each type of icon have a unique graphical image associated with it. In fact, you may want to have several unique images associated with an icon type. For example, you can have several different images associated with the WBTOOL type of icon info file.

Most people will not access the info file directly. The icon manipulation library does all the work needed to read and write info files. The **GetDiskObject()**, **PutDiskObject()**, and **FreeDiskObject()** routines are especially helpful. The calling sequence of each of these is given in the icon library reference pages in the “Library Summaries” appendix.

## THE DISKOBJECT STRUCTURE

The **DiskObject** structure is at the beginning of all info files, and is used in the routines **GetDiskObject()**, **PutDiskObject()**, and **FreeDiskObject()**. The structure is defined in *workbench/workbench.h* and contains the following elements:

### **do\_Magic**

A magic number that the icon library looks for to make sure that the file it is reading really contains an icon. It should be the manifest constant **WB\_DISKMAGIC**. **PutDiskObject()** will put this value in the structure, and **GetDiskObject** will not believe that a file is really an icon unless this value is correct.

### **do\_Version**

This provides a way to enhance the info file in an upwardly-compatible way. It should be **WB\_DISKVERSION**. The icon library will set this value for you and will not believe weird values.

### **do\_Gadget**

This contains all the imagery for the icon. See the “Gadget Structure” section for more details.

### **do\_Type**

The type of the icon (**WBTOOL**, **WBPROJECT**, and so on).

### **do\_DefaultTool**

Default tools are used for projects and disks. For projects the default tool is the program invoked when the project is activated. This tool may be absolute (**DISK:file**), relative to the root of this disk (**:file**), or relative to the project (**file**). If the icon is of type **WBDISK**, the default tool is the diskcopy program that will be used when this disk is the source of a copy.

Note that if the tool is run via the default tool mechanism (for example, a project was activated, not a tool), all the information in the project’s info file is used, and the tool’s info file is ignored. This is especially important for variables like **StackSize** and **ToolWindow**.

### **do\_ToolTypes**

**ToolTypes** is an array of free-format strings. Workbench does not enforce any rules on these strings, but they are useful for passing environment information. See the “ToolTypes” section for more information.

### **do\_CurrentX, do\_CurrentY**

Drawers have a virtual coordinate system. The user can scroll around in this system using the scroll gadgets on the “drawers” window. Each icon in the drawer has a position in the coordinate system. **CurrentX** and **CurrentY** contain the icon’s current position in the drawer.

### **do\_DrawerData**

If the icon is capable of being opened as a drawer (**WBDISK**, **WBDRAWER**, **WBGARBAGE**), it needs a **DrawerData** structure to go with it. This structure contains an Intuition **NewWindow** structure. (see *Amiga Intuition Reference Manual* for more information about windows.) Workbench uses this to hold the current window position and size of the window so it will reopen in the same place. The **CurrentX** and **CurrentY** of the origin of the window is also stored.

### **do\_ToolWindow**

By default, Workbench will start a program without a window. If **ToolWindow** is set,

this file will be opened and made the standard input and output of the program. This window will also be put into the process's `pr_WindowPtr` variable and will be used for all system requesters. Note that this work is actually done in the language-dependent start-up script; if you are coding in assembly language or an unsupported language, you will have to do the work yourself. The only two files that it makes sense to open are `CON:` or `RAW:`. See the AmigaDOS manuals for the full syntax accepted by these devices.

### **do\_StackSize**

This is the size of the stack used for running the tool. If this is null, then Workbench will use a reasonable default stack size (currently 4K bytes).

## **THE GADGET STRUCTURE**

To hold the icon's image, Workbench uses an Intuition **Gadget** structure, defined in `intuition/intuition.h` or `intuition/intuition.i` for the assembly language version. Workbench restricts some of the values of the gadget. Any unused field should be set to 0. For clarity in presentation, you can use the assembly language version of these structures,

*Note:* The C version has the leading "gg\_" stripped off. (Workbench structure members have the same name in all languages supported by Amiga). The Intuition gadget structure members that Workbench pays attention to are listed below:

### **gg\_Width**

This is the width (in pixels) of the active icon's active region. Any mouse button press within this range will be interpreted as having selected this icon.

### **gg\_Height**

The same as **Width**, only in the vertical direction.

### **gg\_Flags**

Currently the gadget *must* be of type `GADGIMAGE`. Three highlight modes are supported: `GADGHCOMP`, `GADGHIMAGE`, and `GADGBACKFILL`. `GADGHCOMP` complements the image specified (as opposed to Intuition, which complements the select box). `GADGHIMAGE` uses an alternate selection image. `GADGBACKFILL` is similar to `GADGHCOMP`, but ensures that there is no "orange ring" around the selected image. It does this by first complementing the image, and then flooding all orange pixels that are on the border of the image to blue. (In case you do not use the default colors, orange is color 3 and blue is color 0.) All other flag bits should be 0.

### **gg\_Activation**

The activation should have only `RELVERIFY` and `GADGIMMEDIATE` set.

### **gg\_Type**

The gadget type should be **BOOLGADGET**.

### **gg\_GadgetRender**

Set this to an appropriate **Image** structure.

### **gg\_SelectRender**

Set this if and only if the highlight mode is **GADGHIMAGE**.

The **Image** structure is typically the same size as the gadget, except that **ig\_Height** is often one pixel less than the gadget height. This allows a blank line between the icon image and the icon name. The image depth *must* be 2; **ig\_PlanePick** *must* be 3; and **ig\_PlaneOnOff** should be 0. The **ig\_NextImage** field should be null.

## **ICONS WITH NO POSITION**

Picking a position for a newly created icon can be tricky. **NO\_ICON\_POSITION** is a magic value for **do\_CurrentX** and **do\_CurrentY** that instructs Workbench to pick a reasonable place for the icon. Workbench will place the icon in an unused region of the drawer. If there is no space in the drawers window, the icon will be placed just to the right of the visible region.

## **Workbench Environment**

When a user activates a tool or project, Workbench runs a program. This program is a separate process and runs asynchronously to Workbench. This allows the user to take advantage of the multiprocessing features of the Amiga.

The environment for a tool under the Workbench is quite different from the environment when a tool is run from the CLI. The CLI does not create a new process for a program; it jumps to the program's code and the program shares the process with the CLI. This means that the program has access to all the CLI's environment, but the program must be very careful to restore all the correct defaults before returning. Workbench starts a tool from scratch and explicitly passes the environment to the tool.

One of the things that a Workbench program must set up is **stdin** and **stdout**. By default, a Workbench program does not have a window to which its output will go. Therefore, **stdin** and **stdout** do not point to legal file handles. If your program attempts to **printf()**, it will destroy the system.

## START-UP MESSAGE

Right after the tool is started, Workbench sends the tool a message, which is posted to the message port in the tool's process. This message contains the environment and the arguments for the tool.

Each icon that is selected in the Workbench is passed to the tool. The first argument is the tool itself. If the tool was derived from a default tool, then this is passed in addition to the project. All other arguments are passed in the order in which the user selected them; the first icon selected will be first.

The tool may do what it wishes with the start-up message; however, it must deallocate the message sooner or later. If the message is replied to Workbench, then Workbench will take care of all the clean-up. The tool should not do this until it finishes executing, because part of the clean-up is freeing the tool's data space.

The start-up message, whose structure is outlined in *workbench/startup.h*, has the following structure elements:

### **sm\_Message**

A standard Exec message. The reply port is set to the Workbench.

### **sm\_Process**

The process descriptor for the tool (as returned by **CreateProcess()**)

### **sm\_Segment**

The loaded code for the tool (returned by **LoadSeg()**)

### **sm\_NumArgs**

The number of arguments in **sm\_ArgList**

### **sm\_ToolWindow**

This is the same string as the **DiskObject's do\_ToolWindow**. It is passed here so the tool's start-up code can open a window for the tool. If it is null, no default window is opened.

### **sm\_ArgList**

This is the argument list itself.

Each argument has two parts. The **wa\_Name** element is the name of the argument. If this is not a default tool or a drawer-like object, this will be the same as the string displayed under the icon. A default tool will have the text of the **do\_DefaultTool** pointer; a drawer will have a null name passed. The **wa\_Lock** is always a lock on a directory, or is NULL (if that object type does not support locks).



The following code fragment will work for all arguments (assuming that open will work on them at all).

```
LockArg(arg)
struct WBArg *arg;
int openmode;
{
 LONG olddir;
 LONG lock;

 /* see if this type can be locked */
 if(arg->wa_Lock == NULL) {
 /* cannot lock it -- it must be a device (for example, DF0:) */
 return(NULL);
 }

 /* change directory to where the argument is */
 olddir = CurrentDir(arg->wa_Lock);

 /* open the argument up */
 lock = Lock(arg->wa_Name, SHARED_LOCK);
 if(lock == NULL) {
 /* who knows: maybe the user canceled a disk insertion
 * request. The real reason can be gotten by IoErr()
 */
 return(NULL);
 }

 /* set the directory back */
 CurrentDir(olddir);

 return(lock);
}
```

For more routines to manipulate Workbench arguments, see the function appendix.

## THE STANDARD START-UP CODE

The standard start-up code handles the worst of the detail work of interfacing with the system. The C start-up code (**startup.obj**) waits for the start-up message, opens the tool window (if one has been requested), sets up **SysBase** and **DOSBase**, and passes the start-up message on to **main()**. When **main()** returns (or **exit()** is called) it replies the message back to Workbench.

The `main()` procedure is called with two parameters: `argv` and `argc`. If `argc` is not NULL, you have been called from the CLI. If `argc` is NULL, you have been called from Workbench. The global variable `WBenchMsg` points to the Workbench start-up message.

*Note:* A word of warning for those of you who do not use the standard start-up sequence: you *must* turn off task switching (with `Forbid()`) before replying the message to Workbench. This will prevent Workbench from unloading your code before you can tell the DOS that you want to exit. See the C start-up code in the “Example Programs” section.

## The ToolTypes Array

This section shows how the `ToolTypes` array should be formatted, and describes the standard entries in the `ToolTypes` array. In brief, `ToolTypes` is an array of strings. These strings can be used to encode information about the icon that will be available to all who wish to use it. The formats are user-definable and user-extensible.

Workbench does not enforce much about the `ToolTypes` array, but some conventions are strongly encouraged. A string may be up to 32K bytes large, but you should not make it over a line long. The alphabet is 8-bit ANSI (for example, normal ASCII with foreign-language extensions). To see what it looks like, try typing with the Alt key held down. Avoid special or non-printing characters. The case of the characters is significant. The general format is

```
<name>=<value>[|<value>]*
```

where `<name>` is the field name and `<value>` is the text to associate with that name. If the ID has multiple values, the values may be separated by a vertical bar. Currently, the value should be the name of the application that understands this file. For example, a basic program might be

```
FILETYPE=ABasiC.program|text
```

This notifies the world that this file is acceptable to either a program that is expecting any arbitrary type of text (for example, an editor) or to a program that only understands a basic program.

Two routines are provided to help you deal with the `Tooltype` array. `FindToolType()` returns the value of a `Tooltype` element. Using the above example, if you are looking for `FILETYPE`, the string “ABasiC.program|text” will be returned.

**MatchToolValue()** returns nonzero if the specified string is in the reference value string. This routine knows how to parse vertical bars. For example, using the reference value string of "ABasiC.program|text", **MatchToolValue()** will return TRUE for "text" and "ABasiC.program" and FALSE for everything else.

## Example Programs

Some example programs, including a start-up sequence, are provided in the following sections.

### FRIENDLYTOOL

This program tells the application if it can understand a particular object.

```
/* INPUTS
 * diskobj -- a workbench DiskObject (a returned by GetDiskObject)
 * id -- the application identifier
 *
 * OUTPUTS
 * nonzero if it understands this object's type
 */

#include "exec/types.h"
#include "workbench/workbench.h"
#include "workbench/icon.h"

LONG IconBase;

FriendlyTool(diskobj, id)
struct DiskObject *diskobj;
char *id;
{
 char **toolarray;
 char *value;

 /* default return value is failure */
 int isfriendly = 0;

 /* this assumes that you have not already opened the icon library
 * elsewhere in your program. You undoubtedly have, because
 * you managed to get a DiskObject structure.
 */
}
```

```

IconBase = OpenLibrary(ICONNAME, 1);
if(IconBase == NULL) {
 /* couldn't find the library??? */
 return(0);
}

/* extract the tool type value array */
toolarray = diskobj->do_ToolType;

/* find the FILETYPE entry */
value = FindToolType(toolarray, "FILETYPE");
if(value) {
 /* info file did define the FILETYPE entry */

 isfriendly = MatchToolValue(value, id);
}

Close(IconBase);

/* protect ourselves from inadvertent use */
IconBase = -1;

return(isfriendly);
}

```

## START-UP PROGRAM

```

*
* C Program Startup/Exit (Combo Version: CLI and WorkBench)
*

***** Included Files *****

INCLUDE "exec/types.i"
INCLUDE "exec/alerts.i"
INCLUDE "exec/nodes.i"
INCLUDE "exec/lists.i"
INCLUDE "exec/ports.i"
INCLUDE "exec/libraries.i"
INCLUDE "exec/tasks.i"
INCLUDE "libraries/dos.i"

```

```
INCLUDE "libraries/dosextens.i"
INCLUDE "workbench/startup.i"
```

```
***** Imported *****
```

```
xlib macro
 xref _LVO1
 endm

 xref _AbsExecBase
 xref _Input
 xref _Output

 xref _main ; C code entry point

 xlib Alert
 xlib FindTask
 xlib Forbid
 xlib GetMsg
 xlib OpenLibrary
 xlib CloseLibrary
 xlib ReplyMsg
 xlib Wait
 xlib WaitPort

 xlib CurrentDir
 xlib Open
```

```
***** Exported *****
```

```
xdef _SysBase
xdef _DOSBase

xdef _errno
xdef _stdin
xdef _stdout
xdef _stderr

xdef _exit ; standard C exit function
xdef _WBenchMsg
```

```
callsys macro
 CALLLIB _LVO1
 endm
```

```

*
* Standard Program Entry Point
*
* main (argc, argv)
* int argc;
* char *argv[];
*

```

```

startup: ; reference for Wack users
 move.l sp,initialSP ; initial task stack pointer
 move.l d0,dosCmdLen
 move.l a0,dosCmdBuf
 clr.l _WBenchMsg

```

```

;----- get Exec's library base pointer:

```

```

 move.l _AbsExecBase,a6
 move.l a6,_SysBase

```

```

;----- get the address of our task

```

```

 suba.l a1,a1
 callsys FindTask
 move.l d0,a4

```

```

;----- are we running as a son of Workbench?

```

```

 tst.l pr_CLI(A4)
 beq fromWorkbench

```

```

;=====
;===== CLI Start-up Code =====
;=====

```

```

fromCLI:

```

```

;----- attempt to open DOS library:

```

```

 bsr openDOS

```

```

;----- find command name:

```

```

 move.l pr_CLI(a4),a0
 add.l a0,a0 ; bcpl pointer conversion
 add.l a0,a0
 move.l cli_CommandName(a0),a0
 add.l a0,a0 ; bcpl pointer conversion
 add.l a0,a0

```

```

;----- create buffer and array:

```

```

* link a6,#-(100+16*4+2*4)
 movem.l d2/a2/a3,-(sp)
 lea argvBuffer,a2
 lea argvArray,a3
* move.l a3,16(sp) ; save
 moveq.l #1,d2 ; param counter

;----- fetch command name:
 moveq.l #0,d0
 move.b (a0)+,d0 ; size of command name
 move.l a2,(a3)+ ; ptr to command name
 bra.s 1$
2$: move.b (a0)+,(a2)+
1$: dbf d0,2$
 clr.b (a2)+

;----- collect parameters:
 move.l dosCmdLen,d0
 move.l dosCmdBuf,a0

;----- skip control characters and space:
3$: move.b (a0)+,d1
 subq.l #1,d0
 ble.s parmExit
 cmp.b #' ',d1
 ble.s 3$

;----- copy parameter:
 addq.l #1,d2
 move.l a2,(a3)+
 bra.s 5$
4$: move.b (a0)+,d1
 subq.l #1,d0
 cmp.b #' ',d1
 ble.s 6$
5$: move.b d1,(a2)+
 bra.s 4$
6$: clr.b (a2)+
 bra.s 3$
parmExit: clr.b (a2)+
 clr.l (a3)+

 move.l d2,d0
 movem.l (sp)+,d2/a2/a3

```

```

 pea argvArray
 move.l d0,-(sp)

```

- \* The above code relies on the end of line containing a control
- \* character of any type, i.e. a valid character must not be the
- \* last. This fact is ensured by DOS.

```

;----- get standard input handle:
 jsr _Input
 move.l d0,_stdin

;----- get standard output handle:
 jsr _Output
 move.l d0,_stdout
 move.l d0,_stderr

;----- call C main entry point
 jsr _main

;----- return success code:
 moveq.l #0,D0
 move.l initialSP,sp ; restore stack ptr
 rts

```

```

;=====
;===== Workbench Start-up Code =====
;=====

```

fromWorkbench:

```

;----- open the DOS library:
 bsr openDOS

;----- we are now set up. wait for a message from our starter
 bsr waitmsg

;----- save the message so we can return it later
 move.l d0,_WBenchMsg

;----- push the message on the stack for wbmain
 move.l d0,-(SP)
 clr.l -(SP) indicate: run from Workbench

;----- get the first argument
 move.l d0,a2
 move.l sm_ArgList(a2),d0

```



```
 beq.s docons
```

```
;----- and set the current directory to the same directory
```

```
 move.l _DOSBase,a6
 move.l d0,a0
 move.l wa_Lock(a0),d1
 callsys CurrentDir
```

```
docons:
```

```
;----- get the toolwindow argument
 move.l sm_ToolWindow(A2),d1
 beq.s domain
```

```
;----- open up the file
 move.l #MODE_OLDFILE,d2
 callsys Open
```

```
;----- set the C input and output descriptors
 move.l d0,_stdin
 move.l d0,_stdout
 move.l d0,_stderr
 beq.s domain
```

```
;----- set the console task (so Open("*" , mode) will work
; waitmsg has left the task pointer in A4 for us
 lsl.l #2,d0
 move.l d0,a0
 move.l fh_Type(a0),pr_ConsoleTask(A4)
```

```
domain:
```

```
 jsr _main
 moveq.l #0,d0 Successful return code
 bra.s exit2
```

```

*
* C Program Exit Function
*
* Warning: this function really needs to do more than this.
*

```

```
_exit:
```

```
 move.l 4(SP),d0 ; extract return code
```

```

exit2:
 move.l initialSP,SP ; restore stack pointer
 move.l d0,-(SP) ; save return code

;----- close DOS library:
 move.l _AbsExecBase,A6
 move.l _DOSBase,d0
 beq.s 1$
 move.l d0,a1
1$: callsys CloseLibrary

;----- if we ran from CLI, skip workbench cleanup:
 tst.l _WBenchMsg
 beq.s exitToDOS

;----- return the startup message to our parent
; we forbid so workbench can't UnLoadSeg() us
; before we are done:
 callsys Forbid
 move.l _WBenchMsg,a1
 callsys ReplyMsg

;----- this rts sends us back to DOS:
exitToDOS:
 move.l (SP)+,d0
 rts

;-----
noDOS:
 ALERT (AG_OpenLib!AO_DOSLib)
 moveq.l #100,d0
 bra.s exit2

;-----
; This routine gets the message that workbench will send to us
; called with task id in A4

waitmsg:
 lea pr_MsgPort(A4),a0 * our process base
 callsys WaitPort
 lea pr_MsgPort(A4),a0 * our process base
 callsys GetMsg
 rts

```

-----  
; Open the DOS library:

openDOS

```
clr.l _DOSBase
lea DOSName,A1
move.l #LIBRARY_VERSION,d0
callsys OpenLibrary
move.l D0,_DOSBase
beq noDOS
rts
```

\*\*\*\*\*

### DATA

\*\*\*\*\*

VerRev dc.w 1,0

\_SysBase dc.l 0

\_DOSBase dc.l 0

\_errno dc.l 0

\_stdin dc.l -1

\_stdout dc.l -1

\_stderr dc.l -1

initialSP dc.l 0

\_WBenchMsg dc.l 0

dosCmdLen dc.l 0

dosCmdBuf dc.l 0

argvArray ds.l 32

argvBuffer ds.b 256

DOSName DOSNAME

END

## ECHO.C

The following example program prints out arguments passed by the CLI or the WorkBench.

```
/* Note: If WB startup, uses window opened by LStartup.obj */

#include <exec/types.h>
#include <workbench/startup.h>
#include <lattice/stdio.h>
extern struct WBStartup *WBenchMsg;

main(argc,argv)
int argc;
char **argv;
{
 BYTE c;
 if(argc>0) {
 printCliArgs(argc,argv);
 }
 else {
 printWBArgs(WBenchMsg);
 while ((c=getchar()) != '\n');
 }
}

printCliArgs(argc,argv)
int argc;
char **argv;
{
 int i;
 for(i=0; i<argc; i++) {
 printf("Arg %2ld = %s\n",i,argv[i]);
 }
}

printWBArgs(msg)
struct WBStartup *msg;
{
 struct WBArg *arg;
 int i;
 for(i=0, arg=msg->sm_ArgList; i < msg->sm_NumArgs; i++,arg++) {
 printf("WBArg%2ld:Lock=0x%06lx:Name=%s\n",
 i,arg->wa_Lock,arg->wa_Name);
 }
 printf("PRESS <RET> TO EXIT\n");
}
```



# Appendix A

## Library Summaries

This appendix contains UNIX-like summaries for the routines that are built into the Amiga ROM (or kickstart) software, as well as summaries of routines in disk-loadable libraries. The debug library documentation is included here as well.

These documentation files are organized alphabetically. Following this introduction is a listing of each routine in this appendix, followed by the name of the library in which the routine is located. The tutorial sections of this manual show you how these routines relate to one another and give you the prerequisites for calling them.

Most routines are listed as part of a library of routines. Before you can use a routine within your program, you must make sure that the library is opened. Opening libraries is explained fully in the "Libraries" chapter of *Amiga ROM Kernel Reference Manual: Exec* but it bears repeating here. You open a library by using the **OpenLibrary()** function as follows:

```
struct LibBase *LibBase;
LibBase = OpenLibrary("library.name",version);
```

where

**library.name**

is a string that describes the name of the library you wish to open.

**version**

is the version number of the library that you wish to have opened. A value of 0 says give me any version. A value of 31, for example (which is the latest version as of this writing) means specifically to open version 31 of this library or a later version if 31 is not available.

If the library is disk-resident, it is loaded and initialized. The **OpenLibrary()** function returns the address of the library base, which you must assign to a specific variable. In this way your program links into the library-specific interface code that is contained in *amiga.lib*.

The names of the libraries that are currently part of the Amiga software and the corresponding names of the library base pointers associated with them are as follows:

| <b>Library Name</b>     | <b>Library Base Pointer Name</b> |
|-------------------------|----------------------------------|
| exec.library            | ExecBase                         |
| clist.library           | ClistBase                        |
| graphics.library        | GfxBase                          |
| layers.library          | LayersBase                       |
| intuition.library       | IntuitionBase                    |
| mathfp.library          | MathBase                         |
| mathtrans.library       | MathTransBase                    |
| mathieeedoubbas.library | MathIeeeDoubBasBase              |
| dos.library             | DosBase                          |
| translator.library      | TranslatorBase                   |
| icon.library            | IconBase                         |
| diskfont.library        | DiskfontBase                     |
| ramlib.library          | ---                              |

(not useful to C language)

For example:

```
#include "graphics/gfx.h"
struct GfxBase *GfxBase;
GfxBase = OpenLibrary("graphics.library",0);
if(GfxBase == NULL) exit(NO_GRAPHICS_LIBRARY_FOUND);
```

*Note:* If your program is coming up through the normal start-up code (see the "Workbench" chapter), *exec.library* and *dos.library* are already opened for you. Thus you need not open them yourself.

The logic of this code is as follows:

1. When calling a routine, C takes the parameters for the routine and pushes them onto the stack. For example:

```
x = Routine(parmA, parmB);
```

Then it calls a routine named "Routine" (adds an underscore to the head of the routine name).

2. The underlying ROM (or disk-based) code usually expects its parameters to be passed in registers rather than on the stack. This is to make the code truly general-purpose (that is, it does not impose a particular stack frame) and more efficient for assembly language coding.

Therefore, the interface code **at Routine**, in turn, saves the contents of registers the routine will use, pulls parameters off the stack, jams them into registers, and finally passes control directly to the actual starting location of the routine itself.

The linker needs the library base location because it is through a "jump-with-offset" from a machine register that the **Routine** entry point is found. The Amiga uses a relocating loader in AmigaDOS, so you can never be sure exactly where a library of routines is located. However, once the system has loaded a library, it knows how and where to find it and gives you a way to use the library's routines.

The following shows typical interface code linked to your program from *amiga.lib*:



```

xref _LibBase ;library base name is defined in
 ;user's file, this code gets linked
 ;to user's program; get the value
 ;from there when library is opened.

xdef _Routine
 ;make _Routine name external,
 ;visible to linker.

_Routine:
 move.l A6,-(sp) ;save register(s)
 move.l 8(sp),A0/A1 ;copy params A and B to regs.
 move.l _LibBase,A6 ;load library base address
 jsr _LVORoutine(A6) ;go to real routine
 move.l (sp)+,A6 ;restore registers
 rts

```

where **\_LVORoutine** is a value representing the offset, within the library, at which the “real” routine (the routine that expects parameters in registers) is located.

When you have finished using a library, at the end of your program, you should close it, using the **CloseLibrary()** function as follows:

```
CloseLibrary(LibBase);
```

If the system is running out of memory and needs to free up space, it can check the library-accessors field for various libraries. For those whose accessors value is zero, it can retrieve the memory that the library had used.

## Contents

-----

|                   |                   |
|-------------------|-------------------|
| abs               | mathffp.library   |
| AddAnimOb         | graphics.library  |
| AddBob            | graphics.library  |
| AddDevice         | exec.library      |
| AddFont           | graphics.library  |
| AddFreeList       | icon.library      |
| AddGadget         | intuition.library |
| AddHead           | exec.library      |
| AddIntServer      | exec.library      |
| AddLibrary        | exec.library      |
| AddPort           | exec.library      |
| AddResource       | exec.library      |
| AddTail           | exec.library      |
| AddTask           | exec.library      |
| AddVSprite        | graphics.library  |
| Allocate          | graphics.library  |
| AllocCList        | exec.library      |
| AllocEntry        | exec.library      |
| AllocMem          | exec.library      |
| AllocRaster       | graphics.library  |
| AllocRemember     | intuition.library |
| AllocSignal       | exec.library      |
| AllocTrap         | exec.library      |
| AllocWBOobject    | icon.library      |
| AndRectRegion     | graphics.library  |
| Animate           | graphics.library  |
| AreaDraw          | graphics.library  |
| AreaEnd           | graphics.library  |
| AreaMove          | graphics.library  |
| AskFont           | graphics.library  |
| AskSoftStyle      | graphics.library  |
| AutoRequest       | intuition.library |
| AvailFonts        | diskfont.library  |
| AvailMem          | exec.library      |
| BeginRefresh      | intuition.library |
| BeginUpdate       | layers.library    |
| BehindLayer       | layers.library    |
| BltBitMap         | graphics.library  |
| BltBitMapRastPort | graphics.library  |
| BltClear          | graphics.library  |
| BltPattern        | graphics.library  |
| BltTemplate       | graphics.library  |
| BuildSysRequest   | intuition.library |
| BumpRevision      | icon.library      |
| Cause             | exec.library      |
| CEND              | graphics.library  |
| ChangeSprite      | graphics.library  |
| CheckIO           | exec.library      |
| CINIT             | graphics.library  |
| ClearDMRequest    | intuition.library |
| ClearEOL          | graphics.library  |
| ClearMenuStrip    | intuition.library |
| ClearPointer      | intuition.library |
| ClearRegion       | graphics.library  |
| ClearScreen       | graphics.library  |
| ClipBlit          | graphics.library  |
| Close             | dos.library       |

|                    |                      |
|--------------------|----------------------|
| CloseDevice        | exec.library         |
| CloseFont          | graphics.library     |
| CloseLibrary       | exec.library         |
| CloseScreen        | intuition.library    |
| CloseWindow        | intuition.library    |
| CloseWorkBench     | intuition.library    |
| CMOVE              | graphics.library     |
| ColdReset          | exec.library         |
| ConcatCList        | clist.library        |
| CopyCList          | clist.library        |
| CopySBitMap        | graphics.library     |
| CreateBehindLayer  | layers.library       |
| CreateDir          | dos.library          |
| CreateExtIO        | exec_support.library |
| CreateProc         | dos.library          |
| CreateStdIO        | exec_support.library |
| CreateUpfrontLayer | layers.library       |
| CurrentDir         | dos.library          |
| CurrentTime        | intuition.library    |
| CWAIT              | graphics.library     |
| DateStamp          | dos.library          |
| Deallocate         | exec.library         |
| Delay              | dos.library          |
| DeleteFile         | dos.library          |
| DeleteLayer        | layers.library       |
| DeletePort         | exec_support.library |
| DeleteStdIO        | exec_support.library |
| DeviceProc         | dos.library          |
| Disable            | exec.library         |
| DisownBlitter      | graphics.library     |
| DisplayAlert       | intuition.library    |
| DisplayBeep        | intuition.library    |
| DisposeLayerInfo   | layers.library       |
| DisposeRegion      | graphics.library     |
| DoCollision        | graphics.library     |
| DoIO               | exec.library         |
| DoubleClick        | intuition.library    |
| Draw               | graphics.library     |
| DrawBorder         | intuition.library    |
| DrawGList          | graphics.library     |
| DrawImage          | intuition.library    |
| DupLock            | dos.library          |
| Enable             | exec.library         |
| EndRefresh         | intuition.library    |
| EndRequest         | intuition.library    |
| EndUpdate          | layers.library       |
| Enqueue            | exec.library         |
| Examine            | dos.library          |
| Execute            | dos.library          |
| Exit               | dos.library          |
| ExNext             | dos.library          |
| faddi              | mathfp.library       |
| FattenLayerInfo    | layers.library       |
| fcmpi              | mathfp.library       |
| fdivi              | mathfp.library       |
| fflti              | mathfp.library       |
| FindName           | exec.library         |
| FindPort           | exec.library         |
| FindTask           | exec.library         |
| FindToolType       | icon.library         |

|                   |                         |
|-------------------|-------------------------|
| Flood             | graphics.library        |
| Forbid            | exec.library            |
| FlushCList        | clist.library           |
| fmul              | mathffp.library         |
| fneg              | mathffp.library         |
| FreeCList         | clist.library           |
| FreeColorMap      | graphics.library        |
| FreeCopList       | graphics.library        |
| FreeCprList       | graphics.library        |
| FreeDiskObject    | icon.library            |
| FreeEntry         | exec.library            |
| FreeFreeList      | icon.library            |
| FreeGBuffers      | graphics.library        |
| FreeMem           | exec.library            |
| FreeRaster        | graphics.library        |
| FreeRemember      | intuition.library       |
| FreeSignal        | exec.library            |
| FreeSprite        | graphics.library        |
| FreeSysRequest    | intuition.library       |
| FreeTrap          | exec.library            |
| FreeVPortCopLists | graphics.library        |
| FreeWBOject       | icon.library            |
| fsub              | mathffp.library         |
| ftsti             | mathffp.library         |
| GetCC             | exec.library            |
| GetCLBuf          | clist.library           |
| GetCLChar         | clist.library           |
| GetCLWord         | clist.library           |
| GetColorMap       | graphics.library        |
| GetDefPrefs       | intuition.library       |
| GetDiskObject     | icon.library            |
| GetGBuffers       | graphics.library        |
| GetIcon           | icon.library            |
| GetMsg            | exec.library            |
| GetPrefs          | intuition.library       |
| GetRGB4           | graphics.library        |
| GetSprite         | graphics.library        |
| GetWBOject        | icon.library            |
| IEEEDPAbs         | mathieeedoubbas.library |
| IEEEDPAdd         | mathieeedoubbas.library |
| IEEEDPCmp         | mathieeedoubbas.library |
| IEEEDPDiv         | mathieeedoubbas.library |
| IEEEDPFlt         | mathieeedoubbas.library |
| IEEEDPMul         | mathieeedoubbas.library |
| IEEEDPNeg         | mathieeedoubbas.library |
| IEEEDPSub         | mathieeedoubbas.library |
| IEEEDPTst         | mathieeedoubbas.library |
| IncrCLMark        | clist.library           |
| Info              | dos.library             |
| InitArea          | graphics.library        |
| InitBitMap        | graphics.library        |
| InitCLPool        | clist.library           |
| InitGels          | graphics.library        |
| InitGMasks        | graphics.library        |
| InitLayers        | layers.library          |
| InitMasks         | graphics.library        |
| InitRastPort      | graphics.library        |
| InitRequester     | intuition.library       |
| InitStruct        | exec.library            |
| InitTmpRas        | graphics.library        |

|                    |                   |
|--------------------|-------------------|
| InitView           | graphics.library  |
| InitVPort          | graphics.library  |
| Input              | dos.library       |
| Insert             | exec.library      |
| IntuiTextLength    | intuition.library |
| IoErr              | dos.library       |
| IsInteractive      | dos.library       |
| ItemAddress        | intuition.library |
| LoadRGB4           | graphics.library  |
| LoadSeg            | dos.library       |
| LoadView           | graphics.library  |
| Lock               | dos.library       |
| LockLayer          | layers.library    |
| LockLayerInfo      | layers.library    |
| LockLayerRom       | graphics.library  |
| LockLayers         | layers.library    |
| MakeLibrary        | exec.library      |
| MakeScreen         | intuition.library |
| MakeVPort          | graphics.library  |
| MarkCList          | clist.library     |
| MatchToolValue     | icon.library      |
| ModifyIDCMP        | intuition.library |
| ModifyProp         | intuition.library |
| Move               | graphics.library  |
| MoveLayer          | layers.library    |
| MoveLayerInFrontOf | layers.library    |
| MoveScreen         | intuition.library |
| MoveSprite         | graphics.library  |
| MoveWindow         | intuition.library |
| MrgCop             | graphics.library  |
| NewLayerInfo       | layers.library    |
| NewRegion          | graphics.library  |
| OffGadget          | intuition.library |
| OffMenu            | intuition.library |
| OnGadget           | intuition.library |
| OnMenu             | intuition.library |
| Open               | dos.library       |
| OpenDevice         | exec.library      |
| OpenDiskFont       | diskfont.library  |
| OpenFont           | graphics.library  |
| OpenLibrary        | exec.library      |
| OpenResource       | exec.library      |
| OpenScreen         | intuition.library |
| OpenWindow         | intuition.library |
| OpenWorkBench      | intuition.library |
| OrRectRegion       | graphics.library  |
| Output             | dos.library       |
| OwnBlitter         | graphics.library  |
| ParentDir          | dos.library       |
| PeekCLMark         | clist.library     |
| Permit             | exec.library      |
| PolyDraw           | graphics.library  |
| PrintIText         | intuition.library |
| PutCLBuf           | clist.library     |
| PutCLChar          | clist.library     |
| PutCLWord          | clist.library     |
| PutDiskObject      | icon.library      |
| PutIcon            | icon.library      |
| PutMsg             | exec.library      |
| PutWBObject        | icon.library      |

|                 |                   |
|-----------------|-------------------|
| QBlit           | graphics.library  |
| QBSBlit         | graphics.library  |
| Read            | dos.library       |
| ReadPixel       | graphics.library  |
| RectFill        | graphics.library  |
| RefreshGadgets  | intuition.library |
| RemakeDisplay   | intuition.library |
| RemDevice       | exec.library      |
| RemFont         | graphics.library  |
| RemHead         | exec.library      |
| RemIBob         | graphics.library  |
| RemIntServer    | exec.library      |
| RemLibrary      | exec.library      |
| Remove          | exec.library      |
| RemoveGadget    | intuition.library |
| RemPort         | exec.library      |
| RemResource     | exec.library      |
| RemTail         | exec.library      |
| RemTask         | exec.library      |
| RemVSprite      | graphics.library  |
| Rename          | dos.library       |
| ReplyMsg        | exec.library      |
| ReportMouse     | intuition.library |
| Request         | intuition.library |
| RethinkDisplay  | intuition.library |
| ScreenToBack    | intuition.library |
| ScreenToFront   | intuition.library |
| ScrollLayer     | layers.library    |
| ScrollRaster    | graphics.library  |
| ScrollVPort     | graphics.library  |
| Seek            | dos.library       |
| SendIO          | exec.library      |
| SetAPen         | graphics.library  |
| SetBPen         | graphics.library  |
| SetOPen         | graphics.library  |
| SetCollision    | graphics.library  |
| SetComment      | dos.library       |
| SetDMRequest    | intuition.library |
| SetDrMd         | graphics.library  |
| SetExcept       | exec.library      |
| SetFont         | graphics.library  |
| SetFunction     | exec.library      |
| SetIntVector    | exec.library      |
| SetMenuStrip    | intuition.library |
| SetPointer      | intuition.library |
| SetProtection   | dos.library       |
| SetRast         | graphics.library  |
| SetRGB4         | graphics.library  |
| SetSignal       | exec.library      |
| SetSoftStyle    | graphics.library  |
| SetSR           | exec.library      |
| SetTaskPri      | exec.library      |
| SetWindowTitles | intuition.library |
| ShowTitle       | intuition.library |
| Signal          | exec.library      |
| SizeCList       | clist.library     |
| SizeLayer       | layers.library    |
| SizeWindow      | intuition.library |
| SortGList       | graphics.library  |
| SPAbs           | mathffp.library   |

|                          |                   |
|--------------------------|-------------------|
| SPAcos                   | mathtrans.library |
| SPAdd                    | mathffp.library   |
| SPAsin                   | mathtrans.library |
| SPAtan                   | mathtrans.library |
| SPCmp                    | mathffp.library   |
| SPCos                    | mathtrans.library |
| SPCosh                   | mathtrans.library |
| SPDiv                    | mathffp.library   |
| SPExp                    | mathtrans.library |
| SPFieee                  | mathtrans.library |
| SPFlt                    | mathffp.library   |
| SplitCList               | clist.library     |
| SPLog                    | mathtrans.library |
| SPLog10                  | mathtrans.library |
| SPMul                    | mathffp.library   |
| SPNeg                    | mathffp.library   |
| SPPow                    | mathtrans.library |
| SPSin                    | mathtrans.library |
| SPSincos                 | mathtrans.library |
| SPSinh                   | mathtrans.library |
| SPSqrt                   | mathtrans.library |
| SPSub                    | mathffp.library   |
| SPTan                    | mathtrans.library |
| SPTanh                   | mathtrans.library |
| SPTieee                  | mathtrans.library |
| SPTst                    | mathffp.library   |
| SubCList                 | clist.library     |
| SumLibrary               | exec.library      |
| SuperState               | exec.library      |
| SwapBitsRastPortClipRect | layers.library    |
| SyncSBitMap              | graphics.library  |
| Text                     | graphics.library  |
| TextLength               | graphics.library  |
| ThinLayerInfo            | layers.library    |
| UnGetCLChar              | clist.library     |
| UnGetCLWord              | clist.library     |
| UnLoadSeg                | dos.library       |
| UnLock                   | dos.library       |
| UnlockLayer              | layers.library    |
| UnlockLayerInfo          | layers.library    |
| UnlockLayerRom           | graphics.library  |
| UnlockLayers             | layers.library    |
| UnPutCLChar              | clist.library     |
| UnPutCLWord              | clist.library     |
| UpfrontLayer             | layers.library    |
| UserState                | exec.library      |
| VBeamPos                 | graphics.library  |
| ViewAddress              | intuition.library |
| ViewPortAddress          | intuition.library |
| Wait                     | exec.library      |
| WaitBlit                 | graphics.library  |
| WaitBOVP                 | graphics.library  |
| WaitForChar              | dos.library       |
| WaitIO                   | exec.library      |
| WaitPort                 | exec.library      |
| WaitTOF                  | graphics.library  |
| WBenchToBack             | intuition.library |
| WBenchToFront            | intuition.library |
| WhichLayer               | layers.library    |
| WindowLimits             | intuition.library |

WindowToBack  
WindowToFront  
Write  
WritePixel  
XorRectRegion

intuition.library  
intuition.library  
dos.library  
graphics.library  
graphics.library



abs

NAME

abs -- obtain the absolute value of the fast floating-point number

C USAGE

```
fnum2 = abs(fnum1);
 D0
```

FUNCTION

Accepts a floating-point number and returns the absolute value of said number. Note that this function is called by compiler-generated code, not by a user-generated function call.

INPUTS

fnum1 - floating-point number

RESULT

fnum2 - floating-point absolute value of fnum1

BUGS

None

SEE ALSO

SPAbs,

AddAnimOb

NAME

AddAnimOb -- add an AnimOb to the linked list of AnimObs

SYNOPSIS

```
AddAnimOb(anOb, anKey, RPort)
 a0 a1 a2
```

FUNCTION

Links this AnimOb into the current list pointed to by animKey  
Initializes all the Timers of the AnimOb's components  
Calls AddBob with each component's Bob  
Note that the RPort must be correctly initialized before you call here,  
including a valid GelsInfo

INPUTS

anOb = pointer to the AnimOb structure to be added to the list  
anKey = address of a ptr to the first AnimOb in the list (NULL if none)  
RPort = pointer to a valid RastPort

RESULT

Nothing

BUGS

None known

SEE ALSO

Nothing

## AddBob

### NAME

AddBob -- adds a Bob to current GEL list

### SYNOPSIS

```
AddBob(Bob, RPort)
 a0 al
```

### FUNCTION

Sets up the system Bob flags, then links this GEL into the list via AdvSprite

### INPUTS

Bob = pointer to the Bob structure to be added to the GEL list  
RPort = pointer to a RastPort structure

### RESULT

Nothing

### BUGS

None known

### SEE ALSO

AdvSprite

## AddDevice

### NAME

AddDevice -- add a device to the system

### SYNOPSIS

```
AddDevice(device)
 Al
```

### FUNCTION

This function adds a new device to the system, making it available to everyone. The device should be ready to be called at this time.

### INPUTS

device - pointer to a properly initialized device node

### SEE ALSO

RemDevice

## AddFont

### NAME

AddFont -- add a font to the system list

### SYNOPSIS

AddFont(textFont), GraphicsLib  
A1 A6

### FUNCTION

This function adds the text font to the system, making it available for use by any application. The font added must be in public memory and must remain until successfully removed.

### INPUTS

textFont - a TextFont structure in public RAM.

## AddFreeList

### NAME

AddFreeList -- add memory to the free list

### SYNOPSIS

status = AddFreeList( free, mem, len )  
D0 A0 A1 A2

### FUNCTION

This routine adds the specified memory to the free list. The free list will be extended (if required). If there is not enough memory to complete the call, a null is returned.

Note that AddFreeList does NOT allocate the requested memory. It only records the memory in the free list.

### INPUTS

free -- a pointer to a FreeList structure  
mem -- the base of the memory to be recorded  
len -- the length of the memory to be recorded

### RESULTS

status -- nonzero if the call succeeded.

### EXCEPTIONS

### SEE ALSO

AllocEntry, FreeEntry, FreeFreeList

### BUGS

## AddGadget

### NAME

AddGadget -- add a gadget to the gadget list of the window or screen

### SYNOPSIS

```
AddGadget(Pointer, Gadget, Position)
 A0 A1 D0
```

### FUNCTION

Adds the specified gadget to the gadget list of the given window, linked in at the position in the list specified by the Position argument (that is, if Position == 0, the gadget will be inserted at the head of the list, and if Position == 1, the gadget will be inserted after the first gadget and before the second). If the Position you specify is greater than the number of gadgets in the list, your gadget will be added to the end of the list. This procedure returns the position at which your gadget was added.

Calling AddGadget() does not cause your gadget to be displayed. The benefit of this is that you may add several gadgets without having the gadget list redrawn every time. The drawback is that you are obliged to call RefreshGadgets() to have your added gadgets displayed.

NOTE: A relatively safe way to add the gadget to the end of the list is to specify a Position of -1. That way, only the 65,536th (and multiples of it) will be inserted at the wrong position. The return value of the procedure will tell you where it was actually inserted.

NOTE: The system window and screen gadgets are initially added to the front of the gadget list. The reason for this is: if you position your own gadgets in some way that interferes with the graphical representation of the system gadgets, the system's gadgets will be "hit" first by the user. If you then start adding gadgets to the front of the list, you will disturb this plan, so beware. On the other hand, if you do not violate the design rule of never overlapping your gadgets, there is no problem.

### INPUTS

Pointer = pointer to the window to get your gadget.  
Gadget = pointer to the new gadget.  
Position = integer position in the list for the new gadget  
(starting from zero as the first position in the list).

### RESULT

Returns the position where the gadget was actually added.

### BUGS

None.

### SEE ALSO

RemoveGadget().

## AddHead

### NAME

AddHead -- insert node at the head of a list

### SYNOPSIS

```
AddHead(list, node)
 A0 A1
```

### FUNCTION

Add a node to the head of a doubly linked list.

### INPUTS

list - a pointer to the target list header  
node - the node to insert at head

## AddIntServer

### NAME

AddIntServer -- add an interrupt server to the system

### SYNOPSIS

```
AddIntServer(intNum, interrupt)
 D0-0:4 A1
```

### FUNCTION

This function adds a new interrupt server to a given server chain. The node is located on the chain in a priority dependent position. Higher priority nodes will be serviced first.

If this server is the first one, interrupt will be enabled on this chain.

Servers are called with the following register conventions:

D0 - scratch  
D1 - scratch

A0 - scratch  
A1 - server data segment pointer (scratch)

A5 - jump vector register (scratch)  
A6 - library base pointer (scratch)

all other registers - must be preserved

### INPUTS

intNum - the Portia interrupt bit (0..14)  
interrupt - pointer to an interrupt server node

### SEE ALSO

RemIntServer

## AddLibrary

### NAME

AddLibrary -- add a library to the system

### SYNOPSIS

```
AddLibrary(library)
 A1
```

### FUNCTION

This function adds a new library to the system making it available to everyone. The library should be ready to be called at this time. It will be added to the system library name list, and the checksum on the library entries will be calculated.

### INPUTS

library - pointer to a properly initialized library structure

### SEE ALSO

RemLibrary

## AddPort

### NAME

AddPort -- add a message port to the system

### SYNOPSIS

```
AddPort(port)
 Al
```

### FUNCTION

This function attaches a message port structure to the system's message port list. The name and priority fields of the port structure should be initialized prior to calling this function. If the user does not require the name and priority fields, they should be initialized to zero. As with the name field in other system list items, the name is useful when more than one task needs to rendezvous on at port.

### INPUTS

port - pointer to a message port

### SEE ALSO

RemPort, FindName

## AddResource

### NAME

AddResource -- add a resource to the system

### SYNOPSIS

```
AddResource(resource)
 Al
```

### FUNCTION

This function adds a new resource to the system and makes it available to other users. The resource should be ready to be called at this time.

### INPUTS

resource - pointer to a properly initialized resource node

### SEE ALSO

RemResource

## AddTail

### NAME

AddTail -- append node to tail of a list

### SYNOPSIS

AddTail(list, node)  
A0 A1

### FUNCTION

Add a node to the tail of a doubly linked list.

### INPUTS

list - a pointer to the target list header  
node - the node to insert at tail

## AddTask

### NAME

AddTask -- add a task to the system

### SYNOPSIS

AddTask(task, initialPC, finalPC)  
A1 A2 A3

### FUNCTION

Add a task to the system.

Certain fields of the task control block must be initialized and a minimal stack should be allocated prior to calling this function.

This function will temporarily use space from the new task's stack for the task's initial set of registers. This space is allocated starting at the SPREG location specified in the task control block (not from SPUPPER). This means that a task's stack may contain static data put there prior to its execution. This is useful for providing initialized global variables or some tasks may want to use this space for passing the task its initial arguments.

A task's initial registers are set to zero (except the PC).

### INPUTS

task - pointer to the task control block  
initialPC - the initial entry point  
finalPC - the finalization code entry point. If zero, the system will use a general finalizer. This pointer is placed on the stack as if it were the outermost return address.

### SEE ALSO

RemTask

## AddVSprite

### NAME

AddVSprite -- add a VSprite to the current GEL list

### SYNOPSIS

AddVSprite(VS, RPort) as called by C  
a0 al

### FUNCTION

Sets up the system VSprite flags  
Links this VSprite into the current GEL list using its Y,X

### INPUTS

VS = pointer to the VSprite structure to be added to the GEL list  
RPort = pointer to a RastPort structure

### RESULT

Nothing

### BUGS

None known

### SEE ALSO

Nothing

## Allocate

### NAME

Allocate -- allocate a block of memory

### SYNOPSIS

memoryBlock = allocate(freeList, byteSize)  
D0 A0 D0

### FUNCTION

This function is used to allocate blocks of memory from a given free memory pool. It will return the first free block that is greater than or equal to the requested size.

All blocks, whether free or allocated, will be block aligned; hence, all allocation sizes are rounded up to the next block even value (e.g. the minimum allocation resolution is 8 bytes).

This function, when used in conjunction with a private free list, can be used to manage an application's internal data memory.

### INPUTS

freeList - points to the memory list header  
byteSize - the size of the desired block in bytes

### RESULT

memoryBlock - a pointer to the just allocated free block.  
If there are no free regions large enough to satisfy the request, return zero. If the amount of requested memory is invalid, return zero.

### EXCEPTIONS

If the free list is corrupt, the system will panic.

### SEE ALSO

Deallocate



## AllocClist

NAME  
AllocClist -- allocate and initialize a clist

SYNOPSIS  
clist = AllocClist(cLPool)  
D0                   A1

FUNCTION  
Get a descriptor that can be used to reference a clist. The clist described is empty. Clists that are no longer in use must be explicitly closed with FreeClist in order to free all their memory: an empty clist still consumes clist pool resources.

INPUTS  
cLPool -  
A clist pool that has already been initialized.

RESULTS  
clist -  
a longword descriptor for a clist that can be used for clist functions.

EXCEPTIONS  
if clist is negative, no space was available for a new clist.

NOTES  
This function is implicitly performed by BufToCL.

## AllocEntry

NAME  
AllocEntry -- allocate many regions of memory

SYNOPSIS  
memList = AllocEntry(memList)  
D0                   A0

FUNCTION  
This routine takes a memList structure and allocates enough memory to hold the required memory as well as a MemList structure to keep track of it. These MemList structures may be linked together in a task control block to keep track of the total memory usage of this task.

INPUTS  
memList -- A memList structure filled in with memEntry structures.

RESULTS  
memList -- A different memList filled in with the actual memory allocated, and their sizes.  
If enough memory cannot be obtained, then the requirements of the failed allocation are returned and bit 31 is set.

EXAMPLES  
The user wants five regions of 2, 4, 8, 16, and 32 bytes in size with requirements of MEMF\_CLEAR, MEMF\_PUBLIC, MEMF\_CHIP.OR.MEMF\_CLEAR, MEMF\_FAST.OR.MEMF\_CLEAR, and MEMF\_PUBLIC.OR.MEMF\_CLEAR respectively. The following code fragment would do that:

```
MemListDecl:
 DS.B LN_SIZE * reserve space for list node
 DC.W 5 * number of entries
 DC.L MEMF_CLEAR * entry #0
 DC.L 2
 DC.L MEMF_PUBLIC * entry #1
 DC.L 4
 DC.L MEMF_CHIP.OR.MEMF_CLEAR * entry #2
 DC.L 8
 DC.L MEMF_FAST.OR.MEMF_CLEAR * entry #3
 DC.L 16
 DC.L MEMF_PUBLIC.OR.MEMF_CLEAR * entry #4
 DC.L 32
```

```
start:
 LEA MemListDecl,A0
 CALLLIB _LVOAllocEntry,A5
```

```
BCLR.L #31,D0
BEQ.S success
```

----- Type of memory that we failed on is in D0

## AllocMem

### NAME

AllocMem -- allocate memory given certain requirements

### SYNOPSIS

```
memoryBlock = AllocMem(byteSize, requirements)
D0 D0 D1-0:31
```

### FUNCTION

This is the memory allocator to be used by system code and applications. It provides a means of specifying whether the allocation should be made in a memory area accessible to the chips, or accessible to shared system code.

The proper allocation of memory is necessary for system code that needs to be compatible with memory mapped systems.

Memory is allocated based on the "requirements" listed. The rule is that (requirements & attributes) == requirements for any particular memory block.

AllocMem will try all memory spaces until one is found with the requested attributes and room for the memory request.

### INPUTS

byteSize - the size of the desired block in bytes  
This number is rounded up to the next larger block size for the actual allocation.  
requirements - (still in flux)  
(see IA\_Structs for bit definitions)

MEMF\_PUBLIC: memory must not be mapped, swapped, or otherwise made non-addressable. ALL MEMORY THAT IS REFERENCED VIA INTERRUPTS AND/OR BY OTHER TASKS MUST BE EITHER PUBLIC OR LOCKED INTO MEMORY! This includes both code and data.

MEMF\_CHIP: Only certain parts of memory are reachable by the special chip sets' DMA circuitry. Anything that will use on-chip DMA must be in memory with this attribute. DMA includes screen memory, things that are blitted, audio blocks, raw disc buffers, etc.

MEMF\_FAST: This is non-chip memory. It is possible for the processor to get locked out of chip memory under certain conditions. If one cannot accept these delays, then one should use FAST memory (by default the system will allocate from FAST memory first anyway).

MEMB\_CLEAR: The memory will be initialized to all zeros.

### RESULT

memoryBlock - a pointer to the allocated free block.  
If there are no free regions large enough to satisfy the request (or if the amount of requested memory is invalid), return zero.

### EXAMPLES

AllocMem(321, MEMB\_CHIP) - private chip memory  
AllocMem(25, MEMB\_PUBLIC) - a "public" system structure that does not require chip memory.

### EXCEPTIONS

If the free list is corrupt, the system will panic.

### SEE ALSO

AllocAbs, FreeMem

## AllocRaster

### NAME

AllocRaster -- allocate space for a Bit Plane

### SYNOPSIS

```
AllocRaster(width, height)
 d0 d1
```

### FUNCTION

This function calls the memory allocation routines to allocate memory space for a bitplane width bits wide and height bits high.

Returns a pointer to the first word if successful.  
Returns 0 if unable to allocate that amount of space.

### INPUTS

x,y are maximum dimensions of the array in bits.

### SEE ALSO

FreeRaster

## AllocRemember

### NAME

AllocRemember -- call AllocMem() and create a link node

### SYNOPSIS

```
AllocRemember(RememberKey, Size, Flags)
 A0 D0 D1
```

### FUNCTION

This routine calls the Exec AllocMem() function for you; it also links the parameters of the allocation into a master list, so that you can simply call the Intuition routine FreeRemember() at a later time to deallocate all allocated memory without being required to remember the details of the memory you have allocated.

This routine has two primary uses:

- o Say that you are doing a long series of allocations in a procedure (such as the Intuition OpenWindow() procedure). If any one of the allocations fails for lack of memory, you need to abort the procedure. Abandoning ship correctly involves freeing up any memory you may have already allocated. This procedure allows you to free up that memory easily, without being required to keep track of how many allocations you have already done, what the sizes of the allocations were, or where the memory was allocated.
- o Also, in the more general case, you may do all of the allocations in your entire program using this routine. Then, when your program is exiting, you can free it all up at once with a simple call to FreeRemember().

You create the "anchor" for the allocation master list by creating a variable that is a pointer to the Remember structure and initializing that pointer to NULL. This is called the RememberKey. Whenever you call AllocRemember(), the routine actually does two memory allocations, one for the memory you want and the other for a copy of a Remember structure. The Remember structure is filled in with data describing your memory allocation, and it is linked into the master list pointed to by your RememberKey. Then, to free up any memory that has been allocated, all you have to do is call FreeRemember() with your RememberKey.

Please read the FreeRemember() function description. As you will see, you can choose to free just the link nodes and keep all the allocated memory for yourself, or you can elect to free both the nodes and your memory buffers.

See this appendix for a description of the AllocMem() call and the values you should use for the Size and Flags variables.

### INPUTS

RememberKey = the address of a pointer to a Remember structure. Before the first call to AllocRemember(), initialize this pointer to NULL. For instance:

```
struct Remember *RememberKey;
```

```
RememberKey = NULL;
AllocRemember(&RememberKey, BUFSIZE, MEMF_CHIP)
FreeRemember(&RememberKey, TRUE)
```

Size = the size in bytes of the memory allocation. Please refer to the Exec AllocMem() function in this appendix for details.

Flags = the specifications for the memory allocation. Please refer to the Exec AllocMem() function in the this appendix for details.

#### RESULT

If the memory allocation is successful, this routine returns the byte address of your requested memory block. Also, the node to your block will be linked into the list pointed to by your RememberKey variable. If the allocation fails, this routine returns NULL and the list pointed to by RememberKey, if any, will be undisturbed.

#### BUGS

None.

#### SEE ALSO

FreeRemember().  
The Exec AllocMem() function.

## AllocSignal

#### NAME

AllocSignal -- allocate a signal bit

#### SYNOPSIS

```
signalNum = AllocSignal(signalNum)
D0 D0
```

#### FUNCTION

Allocate a signal bit from the current tasks pool. Either a particular bit, or the next free bit may be allocated. The signal associated with the newly allocated bit will be properly initialized (cleared).

If the signal is already in use (or no free signals are available) a -1 is returned.

This function can only be used by the currently running task.

#### WARNING

Signals may not be allocated or freed from exception handling code.

#### INPUTS

signalNum - the desired signal number {of 0..31} or -1 for no preference.

#### RESULTS

signalNum - the signal bit number allocated {0..31}.  
If no signals are available, this function returns -1.

#### SEE ALSO

FreeSignal

## AllocTrap

### NAME

AllocTrap -- allocate a processor trap vector

### SYNOPSIS

```
trapNum = AllocTrap(trapNum)
DO DO
```

### FUNCTION

Allocate a trap number from the current task's pool. These trap numbers are those associated with the 68000 TRAP type instructions. Either a particular number, or the next free number may be allocated.

If the trap is already in use (or no free traps are available) a -1 is returned.

This function can only be used by the currently running task.

### WARNING

Signals may not be allocated or freed from exception handling code.

### INPUTS

trapNum - the desired trap number {of 0..15} or -1 for no preference.

### RESULTS

trapNum - the trap number allocated {of 0..15}. If no traps are available, this function returns -1.

### SEE ALSO

FreeTrap

## AllocWBOBJECT

### NAME

AllocWBOBJECT - allocate a Workbench object

### SYNOPSIS

```
object = AllocWBOBJECT()
DO
```

### FUNCTION

This routine allocates a Workbench object and initializes its free list. A subsequent call to FreeWBOBJECT will free all of its memory.

If memory cannot be obtained, a NULL is returned.

This routine is intended only for internal users that can track changes to the Workbench.

### INPUTS

### RESULTS

object - the WBOBJECT (if memory is available)

### EXCEPTIONS

### SEE ALSO

AllocEntry, FreeEntry, FreeWBOBJECT

### BUGS

## AndRectRegion

### NAME

AndRectRegion -- Perform 2d AND operation of rectangle  
with region, leaving result in region

### SYNOPSIS

```
AndRectRegion(region,rectangle)
 a0 a1
```

### Function

Clip away any portion of the region that exists outside  
of the rectangle. Leave the result in region.

### INPUTS

region = pointer to Region structure  
rectangle = pointer to Rectangle structure

### BUGS

## Animate

### NAME

Animate -- processes every AnimOb in the current animation list

### SYNOPSIS

```
Animate(key, RPort)
 a0 a1
```

### FUNCTION

For every AnimOb in the list:

- updates its location and velocities
- calls the AnimOb's special routine if one is supplied
- for each component of the AnimOb
  - if this sequence times out, switches to the new one
  - calls this component's special routine if one is supplied
  - sets the sequence's sprite's y,x coordinates based on all this

### INPUTS

key = address of the variable that points to the head AnimOb  
RPort = pointer to the RastPort structure

### RESULT

Nothing

### BUGS

None known

### SEE ALSO

Nothing

## AreaDraw

### NAME

AreaDraw -- add a point to a list of end points for area-fill.

### SYNOPSIS

```
error = (int) AreaDraw(rp, x, y)
 A1 D0 D1
```

### FUNCTION

Add point to the vector buffer.

### INPUTS

x,y are coordinates of a point in the raster  
rp points to a RastPort structure

### RETURNS

0 if no error  
-1 if no space left in vector list

### SEE ALSO

AreaMove, InitArea, AreaEnd

## AreaEnd

### NAME

AreaEnd -- process table of vectors and produce areafill

### SYNOPSIS

```
error = AreaEnd(rp)
 A1
```

### FUNCTION

Triggers the filling operation.

Processes the vector buffer and generates required fill into the raster planes. After the fill is complete, reinitializes for the next AreaMove. Uses the raster set up by InitTmpRas when generating an areafill mask.

### INPUTS

rp points to a RastPort structure

### RETURNS

0 if no error  
-1 if no space left in vector list

### SEE ALSO

InitArea, AreaMove, AreaDraw

## AreaMove

### NAME

AreaMove -- define a new starting point for a new shape in the vector list

### SYNOPSIS

```
error = AreaMove(rp, x, y)
 A1 D0 D1
```

### FUNCTION

Closes the last polygon and starts another polygon at (x,y). Enters necessary points in vector buffer.

Closing a polygon may result in the generation of another AreaDraw() to close previous polygon.

### INPUTS

x,y are positions in the raster  
rp points to a RastPort structure

### RETURNS

0 if no error  
-1 if no space left in vector list

### SEE ALSO

InitArea, AreaDraw, AreaEnd

## AskFont

### NAME

AskFont -- get the text attributes of the current font

### SYNOPSIS

```
AskFont(rastPort, textAttr), graphicsLib
 A1 A0 A6
```

### FUNCTION

This function fills the text attributes structure with the attributes of the current font in the rastPort.

### INPUTS

rastPort - the RastPort from which the text attributes are extracted  
textAttr - the TextAttr structure to be filled



## AskSoftStyle

### NAME

AskSoftStyle -- get the soft style bits of the current font

### SYNOPSIS

```
enable = AskSoftStyle(rastPort), graphicsLib
 A1 A6
```

### FUNCTION

This function returns those style bits of the current font that are not intrinsic in the font itself but are algorithmically generated. These are the bits that are valid to set in the enable mask for SetSoftStyle

### INPUTS

rastPort - the RastPort from which the font and style are extracted

### RESULTS

enable - those bits in the style algorithmically generated. Style bits that are not defined are also set.

## AutoRequest

### NAME

AutoRequest -- automatically build and get response from a requester

### SYNOPSIS

```
AutoRequest(Window, BodyText, PositiveText, NegativeText,
 A0 A1 A2 A3
 PositiveFlags, NegativeFlags, Width, Height)
 D0 D1 D2 D3
```

### FUNCTION

This procedure automatically builds a requester for you and then waits for a response from the user or the system to satisfy your request. If the response is positive, this procedure returns TRUE. If the response is negative, this procedure returns FALSE.

This procedure first preserves the state of the IDCMP values of the window argument. Then it creates an IDCMPFlag specification by merging your PositiveFlags, NegativeFlags, and the IDCMP class GADGETUP. You may choose to specify no flags for either the PositiveFlags or NegativeFlags arguments.

The IntuiText arguments and the Width and Height values are passed directly to the BuildSysRequest() procedure, along with your window pointer and the IDCMP flags. Please refer to BuildSysRequest() for a description of the IntuiText that you are expected to supply when calling this routine. It is an important but long-winded description that need not be duplicated here.

If the BuildSysRequest() procedure does not return a pointer to a window, it will return TRUE or FALSE (not valid structure pointers) instead, and these BOOL values will be returned immediately.

On the other hand, if a valid window pointer is returned, that window will have had its IDCMP ports and flags initialized according to your specifications. AutoRequest() then waits for an IDCMP message on the UserPort; this message will satisfy one of three requirements:

- o If the message is of a class that matches one of your PositiveFlags arguments (if you have supplied any), this routine returns TRUE.
- o If the message class matches one of your NegativeFlags arguments (if you have supplied any), this routine returns FALSE.
- o The only other possibility is that the IDCMP message is of class GADGETUP, which means that one of the two gadgets, as specified by the PositiveText and NegativeText arguments, was selected by the user. If the TRUE gadget was selected, TRUE is returned. If the FALSE gadget was selected, FALSE is returned.

When the dust has settled, this routine calls FreeSysRequest(), if necessary, to clean up the requester and any other allocated memory.

#### INPUTS

Window = pointer to a Window structure.  
BodyText = pointer to an IntuiText structure.  
PositiveText = pointer to an IntuiText structure.  
NegativeText = pointer to an IntuiText structure.  
PositiveFlags = flags for the IDCMP.  
NegativeFlags = flags for the IDCMP.  
Width, Height = the sizes required for the rendering of the requester.

#### RESULT

The return value is either TRUE or FALSE. See the text above for a complete description of the chain of events that might lead to either of these values being returned.

#### BUGS

None.

#### SEE ALSO

BuildSysRequest().

## AvailFonts

#### NAME

AvailFonts - build an array of all fonts in memory / on disk

#### SYNOPSIS

```
error = AvailFonts(buffer, bufBytes, types);
 A0 D0 D1
```

#### FUNCTION

AvailFonts fills a user supplied buffer with the structure, described below, that contains information about all the fonts available in memory and/or on disk. Those fonts available on disk need to be loaded into memory and opened via OpenDiskFont(); those already in memory are accessed via OpenFont. The TextAttr structure required by the open calls is part of the information AvailFonts() supplies.

#### INPUTS

buffer - memory to be filled with struct AvailFontsHeader followed by an array of AvailFonts elements, which contains entries for the available fonts and their names.

bufBytes - the number of bytes in the buffer

types - AFF\_MEMORY is set to search memory for fonts to fill the structure, AFF\_DISK is set to search the disk for fonts to fill the structure. Both can be specified.

#### RESULTS

buffer - filled with struct AvailFontsHeader followed by the AvailFonts elements, There will be duplicate entries for fonts found both in memory and on disk, differing only by type. The existence of a disk font in the buffer indicates that it exists as an entry in a font contents file -- the underlying font file has not been checked for validity, thus an OpenDiskFont() of it may fail.

error - if non-zero, this indicates the number of bytes needed for AvailFonts in addition to those supplied. Thus structure elements were not returned because of insufficient bufBytes.

## AvailMem

### NAME

AvailMem -- memory available given certain requirements

### SYNOPSIS

```
size = AvailMem(requirements)
D0 D1
```

### FUNCTION

This function returns the size of memory given certain requirements.

### INPUTS

requirements - a requirements mask as specified in AllocMem

### RESULT

size - total free space remaining

## BeginUpdate

### NAME

BeginUpdate -- prepare to repair damaged layer

### SYNOPSIS

```
BeginUpdate(l)
 a0
```

### INPUTS

l = pointer to a layer

### FUNCTION

Converts damage list to ClipRect list and swaps in for programmer to redraw through. This routine simulates the ROM library environment. The layer is locked against changes made by the layer library.

### SEE ALSO

layers.h EndUpdate()

## BeginRefresh

### NAME

BeginRefresh -- set up a window for optimized refreshing

### SYNOPSIS

```
BeginRefresh(Window)
 A0
```

### FUNCTION

This routine sets up your window for optimized refreshing. It sets Intuition internal states and then sets up the layer underlying your window for a call to the layer library. There, the "clip rectangles" of the layer are reorganized in a fashion that causes any drawing performed in your window (until you call EndRefresh()) to occur only in the regions that need to be refreshed. The term "clip rectangles" refers to the division of your window into visible and concealed rectangles. For more information about clipping rectangles and the layer library, refer to the main chapters of this manual.

For instance, if you have a SIMPLE\_REFRESH window that is partially concealed and the user brings it to the front, your program will receive a message asking it to refresh its display. If your program calls BeginRefresh() before doing any of the drawing, the layer that underlies your window will be arranged such that the only drawing that will actually take place will be that which goes to the newly revealed areas. This is very performance-efficient.

After your program has performed its refresh of the display, it should call EndRefresh() to reset the state of the layer and the window. Then the program may proceed with drawing to the window as usual.

Your program learns that the window needs refreshing by receiving either a message of class REFRESHWINDOW through the IDCMP or an input event of class IECLASS\_REFRESHWINDOW through the console device. Whenever the program is told that the window needs refreshing, it should call BeginRefresh() and EndRefresh() to clear the refresh-needed state, even if no drawing will be done.

### INPUTS

Window = pointer to the Window structure that needs refreshing.

### RESULT

None.

### BUGS

None.

### SEE ALSO

EndRefresh().

## BehindLayer

### NAME

BehindLayer -- put layer behind other layers.

### SYNOPSIS

```
BehindLayer(li, l)
 a0 a1
```

### INPUTS

li = pointer to LayerInfo structure  
l = pointer to a layer

### FUNCTION

Moves this layer behind all others, swapping bits in and out of the display with other layers. If other layers are REFRESH, collects their damage lists and sets bit in Flags of those layers that may be revealed. If this layer is a backdrop layer, puts it behind all other backdrop layers. If this layer is NOT a backdrop layer, puts it in front of the top backdrop layer and behind all other layers.

### RETURNS

TRUE if operation successful  
FALSE if operation unsuccessful (probably out of memory)

### BUGS

### SEE ALSO

layers.h

## BltBitMap

### NAME

BltBitMap -- move a rectangle in a raster

### SYNOPSIS

```
planes = BltBitMap(SrcBitMap, SrcX, SrcY, DestBitMap,
D0 A0 D0 D1 A1
 DestX, DestY, SizeX, SizeY, Minterm, Mask, TempA);
D2 D3 D4 D5 D6 D7 A2
```

### FUNCTION

Performs non-destructive blits to move a rectangle from one area in a raster to another area, which can be on a different raster.

### INPUTS

SrcBitMap, DestBitMap - the BitMap(s) containing the rectangles

- the planes copied from the source to the destination are only those whose plane numbers are identical and less than the minimum plane count and whose write mask is non-zero.
- SrcBitMap and DestBitMap can be identical

SrcX, SrcY - the x and y coordinates of the upper left corner of the source rectangle. Valid range is positive signed integer such that the raster word's offset 0..(32767-Size)

DestX, DestY - the x and y coordinates of the upper left corner of the destination for the rectangle. Valid range is as for Src.

SizeX, SizeY - the size of the rectangle to be moved. Valid range is (X: 1..976; Y: 1..1023 such that final raster word's offset is 0..32767)

Minterm - the logic function to apply to the rectangle when A is non-zero (i.e. within the rectangle). B is the source rectangle and C, D is the destination for the rectangle.

- \$0C0 is a vanilla copy
- \$030 inverts the source before the copy
- \$050 ignores the source and inverts the destination
- see the Amiga Hardware Reference Manual for other combinations.

Mask - the write mask to apply to this operation. Bits set indicate the corresponding planes (if not greater than the minimum plane count) are to participate in the operation. Typically, this is set to 0xff.

TempA - If the copy overlaps exactly to the left or right (i.e., the scan line addresses overlap), and TempA is non-zero, it points to enough chip-accessible memory to hold a line of A source for the blit.

### RESULTS

planes - the number of planes actually involved in the blit.

### EXCEPTIONS

This blt is assumed to be friendly: no errors conditions (e.g., a rectangle outside the BitMap bounds) are tested or reported. A plane count that is less than expected can be attributed to a failure to allocate a TempA when it was needed.

## BltBitMapRastPort

### NAME

BltBitMapRastPort -- blit from source bitmap to destination rastport

### SYNOPSIS

```
BltBitMapRastPort
*(srcbm,srcx,srcy,destrp,destX,destY,sizeX,sizeY,minterm)
*a0 d0 d1 a1 d2 d3 d4 d5 d6
```

### FUNCTION

Blits from source bitmap to position specified in destination rastport using minterm.

### INPUTS

srcbm - a pointer to the source bitmap

srcx - x offset into source bitmap

srcy - y offset into source bitmap

destrp - a pointer to the destination rastport

destX - x offset into dest rastport

destY - y offset into dest rastport

sizeX - width of blit in pixels

sizeY - height of blit in rows

minterm - minterm to use for this blit

### RETURNS

TRUE - if blit successfully completed

FALSE - if blit failed

### BUGS

### SEE ALSO

## BltClear

### NAME

BltClear -- clear a block of memory words to zero.

### SYNOPSIS

```
BltClear(memBlock, bytecount, flags)
 al d0 d1
```

### FUNCTION

For memory that is local and blitter accessible. The most efficient way to clear a range of memory locations is to use the system's most efficient data mover, the blitter. This command accepts the starting location and count and clears that block to zeros.

### INPUTS

memBlock pointer to local memory to be cleared  
memBlock must be even  
flags set bit 0 to force function to wait until blit is done.  
set bit 1 to use row/bytesperrow  
bytecount if (flags & 2) == 0 then  
even number of bytes to clear.  
else  
low 16 bits is taken as number of bytes  
per row and upper 16 bits taken as  
number of rows.

This function is somewhat hardware-dependent. In the rows/bytesperrow mode, rows must be  $\leq 1024$  and bytesperrow must be  $\leq 128$ . In standard bytecount mode multiple runs of the blitter may be used to clear all the memory.

### RESULT

The block of memory is set to zeros.

### BUGS

None known.

### SEE ALSO

## BltPattern

### NAME

BltPattern -- Using standard drawing rules for areafill, blit through a mask

### SYNOPSIS

```
BltPattern(RastPort *,char *, xl, yl, maxx, maxy, bytecnt)
 al, a0 d0 d1 d2 d3 d4
```

### FUNCTION

Blit using drawmode,areafill pattern,outline, mask pointed to by a0, at position rectangle (xl,yl) (maxx,maxy). The image is not shifted but must be word aligned.

### INPUTS

al points to RastPort  
a0 points to 2 dimensional mask if needed  
xl,yl upper left of rectangular region in RastPort  
maxx,maxy points to lower right of rectangular region in RastPort  
bytecnt number of BytesPerRow for char \* a0.

### RETURNS

### SEE ALSO

## BltTemplate

### NAME

BltTemplate -- cookie cut a shape in a rectangle to the RastPort

### SYNOPSIS

```
BltTemplate(source, srcX, srcMod, destRastPort,
 A0 D0 D1 A1
 destX, destY, sizeX, sizeY), graphicsLib
 D2 D3 D4 D5
```

### FUNCTION

This function draws the image in the template into the RastPort in the current color and drawing mode at the specified position. The template is assumed not to overlap the destination.

### EXCEPTIONS

If the template falls outside the RastPort boundary, it is truncated to that boundary.

## BuildSysRequest

### NAME

BuildSysRequest -- build and display a system requester

### SYNOPSIS

```
BuildSysRequest(Window, BodyText, PositiveText, NegativeText,
 A0 A1 A2 A3
 IDCMPFlags, Width, Height)
 D0 D1 D2
```

### FUNCTION

This procedure builds a requester based on the supplied information. If all goes well and the requester is constructed, this procedure returns a pointer to the window in which the requester appears. That window will have the IDCMP UserPort and WindowPort initialized to reflect the flags found in the IDCMPFlags argument. The program may then Wait() on those ports to detect the user's response to your requester, which may include either selecting one of the gadgets or causing some other event to be noticed by Intuition (such as DISKINSERTED, for instance). After the requester is satisfied, your program should call the FreeSysRequest() procedure to remove the requester and free any allocated memory.

If it is not possible to construct the requester, this procedure will use the text arguments to construct a text string for a call to the DisplayAlert() procedure and then will return either TRUE or FALSE depending on whether DisplayAlert() returned FALSE or TRUE, respectively.

If the Window argument you supply is equal to NULL, a new window will be created for you in the Workbench screen. If you want the requester created by this routine to be bound to a particular window, you should not supply a Window argument of NULL.

The text arguments are used to construct the display. They are pointers to instances of the IntuiText structure.

The BodyText argument should be used to describe the nature of the requester. As usual with IntuiText data, you may link several lines of text together, and the text may be placed in various locations in the requester. This IntuiText pointer will be stored in the ReqText variable of the new requester.

The PositiveText argument describes the text that you want associated with the user choice of "Yes," "TRUE," "retry," or "good." If the requester is successfully opened, this text will be rendered in a gadget in the lower left of the requester; this gadget will have the GadgetID field set to TRUE. If the requester cannot be opened and the DisplayAlert() mechanism is used, this text will be rendered in the lower left corner of the alert display with additional text specifying that the left mouse button will select this choice. This pointer can be set to NULL, which specifies

that there is no TRUE choice that can be made.

The NegativeText argument describes the text that you want associated with the user choice of "No," "FALSE," "cancel," or "bad." If the requester is successfully opened, this text will be rendered in a gadget in the lower right of the requester; this gadget will have the GadgetID field set to FALSE. If the requester cannot be opened and the DisplayAlert() mechanism is used, this text will be rendered in the lower right corner of the alert display with additional text specifying that the right mouse button will select this choice. This pointer cannot be set to NULL. There must always be a way for the user to cancel this requester.

The positive and negative gadgets created by this routine have the following features:

- o BOOLGADGET
- o RELVERIFY
- o REQGADGET
- o TOGGLESELECT

When defining the text for your gadgets, you may find it convenient to use the special definitions used by Intuition for the construction of the gadgets. These definitions include AUTODRAWMODE, AUTOLEFTEDGE, AUTOTOPEdge and AUTOFRONTPEN. You can find these in your local intuition.h (or intuition.i) file.

The Width and Height values describe the size of the requester. All of your BodyText must fit within the Width and Height of your requester. The gadgets will be created to conform to your sizes.

IMPORTANT NOTE: For the preliminary release of this procedure, a new window is opened in the same screen as the one containing your window. However, with a forthcoming update of Intuition this will change; the requester will be opened in the window supplied as an argument to this routine, if possible. The primary implication of this will be that the IDCMP flags and ports will be disturbed by a call to this routine. To assure upward compatibility, it is your responsibility to make sure that the ports and IDCMPFlags of the window passed to the routine are protected before the call to this routine.

#### INPUTS

Window = pointer to a Window structure.  
BodyText = pointer to an IntuiText structure.  
PositiveText = pointer to an IntuiText structure.  
NegativeText = pointer to an IntuiText structure.  
IDCMPFlags = the IDCMP flags you want used for the initialization of the IDCMP of the window containing this requester.  
Width, Height = the size required to draw your requester.

#### RESULT

If the requester was successfully drawn in a window, the value returned by this procedure is a pointer to the window in which the requester was drawn. If, however, the requester cannot be drawn in the window, this routine will have

called DisplayAlert() before returning and will pass back TRUE if the user pressed the left mouse button and FALSE if the user pressed the right mouse button.

#### BUGS

This procedure currently opens a window and then opens the requester within that window. Also, if DisplayAlert() is called, the PositiveText and NegativeText are not rendered in the lower corners of the alert.

#### SEE ALSO

FreeSysRequest(), DisplayAlert(), ModifyIDCMP(), Wait(), AutoRequest()



## BumpRevision

### NAME

BumpRevision -- reformat a name for a second copy

### SYNOPSIS

```
result = BumpRevision(newbuf, oldname)
DO A0 A1
```

### FUNCTION

BumpRevision takes a name and turns it into a "copy of name." It knows how to deal with copies of copies. The routine will truncate the new name to the maximum DOS name size (currently 30 characters).

### INPUTS

newbuf - the new buffer that will receive the name (it must be at least 31 characters long).  
oldname - the original name

### RESULTS

result - a pointer to newbuf

### EXCEPTIONS

### EXAMPLE

| <u>oldname</u>                   | <u>newbuf</u>                    |
|----------------------------------|----------------------------------|
| "foo"                            | "copy of foo"                    |
| "copy of foo"                    | "copy 2 of foo"                  |
| "copy 2 of foo"                  | "copy 3 of foo"                  |
| "copy 199 of foo"                | "copy 200 of foo"                |
| "copy foo"                       | "copy of copy foo"               |
| "copy 0 of foo"                  | "copy 1 of foo"                  |
| "012345678901234567890123456789" | "copy of 0123456789012345678901" |

### SEE ALSO

### BUGS

## Cause

### NAME

Cause -- cause a software interrupt

### SYNOPSIS

```
Cause(interrupt)
A1
```

### FUNCTION

This function causes a software interrupt to occur. If it is called from user mode (and processor level 0), the software interrupt will preempt the current task.

Currently only 5 software interrupt priorities are implemented: -32, -16, 0, +16, and +32. Priorities in between these values are truncated. Priorities outside the -32/+32 range are not allowed.

### INPUTS

interrupt - pointer to a properly initialized interrupt node

CEND

NAME

CEND -- terminate user Copper list.

SYNOPSIS

CEND( c )

FUNCTION

Adds instruction to terminate user Copper list.

INPUTS

c = pointer to UCopList structure

RESULTS

This is actually a macro that calls CWait(c) to wait for the end of the user Copper list and then calls CBump(c) to bump the local pointer to the next instruction.

BUGS

None Known

SEE ALSO

CINIT();  
CMOVE();  
CWAIT();

ChangeSprite

NAME

ChangeSprite -- change the sprite image pointer.

SYNOPSIS

ChangeSprite( vp, s, newdata)  
a0 al a2

FUNCTION

The sprite image is changed to use the data starting at newdata

INPUTS

vp = pointer to ViewPort structure that this sprite is relative to.  
or 0 if relative only top of View  
s = pointer to SimpleSprite structure  
newdata = pointer to data structure of the following form:

```
struct spriteimage
{
 UWORD posctl[2]; /* used by simple sprite machine*/
 UWORD data[height][2]; /* actual sprite image */
 UWORD reserved[2]; /* initialized to */
 /* 0xFFFF,0xFF7F */
}
```

Programmer must initialize reserved[2]. The spriteimage must be in CHIP memory. The height subfield of the SimpleSprite structure must be set to reflect the height of the new spriteimage BEFORE calling ChangeSprite. The programmer may allocate two sprites to handle a single attached sprite. After GetSprite, ChangeSprite, the programmer can set the SPRITE\_ATTACHED bit in posctl[1] of the odd-numbered sprite.

RESULTS

BUGS

SEE ALSO

sprite.h FreeSprite ChangeSprite MoveSprite

## CheckIO

### NAME

CheckIO -- get the IO request status

### SYNOPSIS

```
result = CheckIO(iORequest)
D0 A1
```

### FUNCTION

This function determines the current state of an I/O request and returns FALSE if the I/O has not yet completed. This function effectively hides the internals of the I/O completion mechanism.

If the I/O has completed, CheckIO will not remove the returned IORequest from the reply port. This should be performed with Remove.

This function SHOULD NOT be used to busy loop, waiting for an IO to complete.

### INPUTS

iORequest - pointer to an I/O request block

### RESULTS

result - null if I/O is still in progress. Otherwise D0 points to the IORequest block.

## CINIT

### NAME

CINIT -- initialize user Copper list to accept intermediate user Copper instructions

### SYNOPSIS

```
struct CopperList *CINIT(c , n)
```

### FUNCTION

allocates/initializes Copper list data structures/buffers

### INPUTS

c = pointer to UCopList structure  
n = number of instructions buffer must hold

### RESULTS

this is actually a macro that calls UCopperListInit(c,n)  
If ( c== 0) allocate CopperList structure and a buffer to hold n Copper instructions. If (c != 0) then just reinitialize the list to accept Copper instructions and ignore n.

### BUGS

## ClearDMRequest

### NAME

ClearDMRequest -- clear the DMRequest of the window

### SYNOPSIS

```
ClearDMRequest(Window)
A0
```

### FUNCTION

Attempts to clear the DMRequester from the specified window. The DMRequester is the special requester that you attach to the double-click of the menu button; the user can then bring up that requester on demand. This routine will not clear the DMRequester if it is active (in use by the user). If you want to change the DMRequester after having called SetDMRequest(), the correct way to start is by calling ClearDMRequest() until it returns a value of TRUE; then you can call SetDMRequest() with the new DMRequester.

### INPUTS

Window = pointer to the structure of a window from which the DMRequest is to be cleared.

### RESULT

If the DMRequest was not currently in use, this function zeroes out the DMRequest pointer in the window and returns TRUE.

If the DMRequest was currently in use, this function does not change the pointer and returns FALSE.

### BUGS

None.

### SEE ALSO

SetDMRequest().  
Request().

## ClearEOL

### NAME

ClearEOL -- clear from current position to end of line

### SYNOPSIS

```
ClearEOL(rastPort), graphicsLib
A1 A6
```

### FUNCTION

Clears a rectangular swath from the current position to the right edge of the rastPort. The height of the swath is taken from that of the current text font, and the vertical positioning of the swath is adjusted by the text baseline, such that text output at this position would lie wholly on this newly cleared area.

Clearing consists of setting the color of the swath to zero, or, if the DrawMode is 2, to the BgPen.

## ClearMenuStrip

NAME ClearMenuStrip -- clear the menu strip from the window

SYNOPSIS  
ClearMenuStrip(Window)  
A0

FUNCTION  
Clears the menu strip from the window.

INPUTS  
Window = pointer to a Window structure.

RESULT  
None.

BUGS  
None.

SEE ALSO  
SetMenuStrip().

## ClearPointer

NAME ClearPointer -- clear the pointer definition from a window

SYNOPSIS  
ClearPointer(Window)  
A0

FUNCTION  
Clears the window of its own definition of the Intuition pointer. After ClearPointer() is called, every time this window is active the default Intuition pointer will be the pointer displayed to the user. If your window is active when this routine is called, the change will take place immediately.

INPUTS  
Window = pointer to the structure of the window to be cleared of its pointer definition.

RESULT  
None.

BUGS  
None.

SEE ALSO  
SetPointer().

## ClearRegion

### NAME

ClearRegion -- set this region to size 0

### SYNOPSIS

```
ClearRegion(region)
 a0
```

### Function

Clip away all rectangles in the region, leaving nothing.

### INPUTS

region = pointer to Region structure

### BUGS

## ClearScreen

### NAME

ClearScreen -- clear from current position to end of RastPort

### SYNOPSIS

```
ClearScreen(rastPort), graphicsLib
 A1 A6
```

### FUNCTION

Clears a rectangular swath from the current position to the right edge of the rastPort with ClearEOL, then clears the rest of the screen from just beneath the swath to the bottom of the rastPort.

Clearing consists of setting the color of the swath to zero, or, if the DrawMode is 2, to the BgPen.

## ClipBlit

### NAME

ClipBlit -- Calls BltBitMap() after accounting for windows

### SYNOPSIS

```
ClipBlit(Src, SrcX, SrcY, Dest, DestX, DestY, XSize, YSize, Minterm);
 a0 d0 d1 a1 d2 d3 d4 d5 d6
```

### FUNCTION

Performs the same function as BltBitMap(), except that it takes into account the Layers and ClipRects of the layer library, all of which are (and should be) transparent to you. So, whereas BltBitMap() requires pointers to BitMaps, ClipBlit requires pointers to the RastPorts that contain the Bitmaps, Layers, et cetera.

If you are going to blit blocks of data around via the RastPort of your Intuition Window, you must call this routine (rather than BltBitMap()).

Either the Src RastPort, the Dest RastPort, both, or neither, can have Layers. This routine takes care of all cases.

See BltBitMap() for a thorough explanation.

### INPUTS

Src = pointer to the RastPort of the source for your blit  
SrcX, SrcY = the topleft offset into Src for your data  
Dest = pointer to the RastPort to receive the blitted data  
DestX, DestY = the topleft offset into the destination RastPort  
XSize = the width of the blit  
YSize = the height of the blit  
Minterm = the boolean blitter function, where SRCB is associated with the Src RastPort and SRCC goes to the Dest RastPort

### RESULT

None

### BUGS

None

### SEE ALSO

BltBitMap();

## Close

### NAME

Close -- close a file for input or output

### SYNOPSIS

```
Close(file)
 Dl
```

### FUNCTION

The file handle 'file' indicates the file that Close should close. You obtain this file handle as a result of a call to Open. You must remember to close explicitly all the files you open in a program. However, you should not close inherited file handles opened elsewhere.

### INPUTS

file - BCPL pointer to a file handle

## CloseDevice

### NAME

CloseDevice -- conclude access to a device

### SYNOPSIS

```
CloseDevice(iORequest)
 A1
```

### FUNCTION

This function informs the system that access to a device/unit previously opened has been concluded. The device may perform certain house-cleaning operations. The I/O request structure is now free to be recycled.

### INPUTS

iORequest - pointer to an I/O request structure

### SEE ALSO

OpenDevice

## CloseFont

### NAME

CloseFont -- release a pointer to a system font.

### SYNOPSIS

```
CloseFont(font), GraphicsLib
 A1 A6
```

### FUNCTION

This function indicates that the font specified is no longer in use. It is used to close a font opened by OpenFont, so that fonts that are no longer in use do not consume system resources.

### INPUTS

font -

A font, as returned by OpenFont.



## CloseLibrary

### NAME

CloseLibrary -- conclude access to a library

### SYNOPSIS

CloseLibrary(library)  
A1

### FUNCTION

This function informs the system that access to the given library has been concluded. The user should not reference the library or any routine in the library after this close.

### INPUTS

library - pointer to a library node

### SEE ALSO

OpenLibrary

## CloseScreen

### NAME

CloseScreen -- close an Intuition screen

### SYNOPSIS

CloseScreen(Screen)  
A0

### FUNCTION

This function unlinks the screen, unlinks the ViewPort, and deallocates everything. It does not care whether or not there are still any windows attached to the screen and does not try to close any attached windows; in fact, it ignores them altogether. If this is the last screen, this function attempts to reopen Workbench.

### INPUTS

Screen = pointer to the Screen structure to be cleared and deallocated.

### RESULT

None.

### BUGS

None.

### SEE ALSO

OpenScreen().

## CloseWindow

### NAME

CloseWindow -- close an Intuition window

### SYNOPSIS

```
CloseWindow(Window)
 A0
```

### FUNCTION

This function closes an Intuition window. It unlinks it from the system, unallocates its memory, and, if its screen is a system one that would be empty without the window, closes the system screen, too.

Caution: if you are ever rude enough to CloseWindow() on a window that has an IDCMP without first having Reply()'d to all of the messages to the IDCMP port, Intuition in turn will be so rude as to reclaim and deallocate its messages without waiting for your permission.

Caution: if you have added a menu strip to this window (via a call to SetMenuStrip()) you must be sure to remove that menu strip (via a call to ClearMenuStrip()) before closing your window. CloseWindow() does not check whether the menus of your window are currently being used when the window is closed. If this happens to be the case, as soon as the user releases the menu button the system will crash.

### INPUTS

Window = a pointer to a Window structure.

### RESULT

None.

### BUGS

None.

### SEE ALSO

OpenWindow(), CloseScreen()

## CloseWorkBench

### NAME

CloseWorkBench -- close the Workbench screen

### SYNOPSIS

```
BOOL CloseWorkBench()
```

### FUNCTION

This routine attempts to close the Workbench. If the Workbench is open, it tests whether or not any applications have opened windows on the Workbench and returns FALSE if so. Otherwise, it cleans up all special buffers, closes the Workbench screen, makes the Workbench program mostly inactive (it will still monitor disk activity), and returns TRUE.

If the Workbench screen isn't open when this routine is called, TRUE is returned immediately.

### INPUTS

None.

### RESULT

TRUE if the Workbench screen is closed.  
FALSE if anything went wrong and the Workbench screen is still out there.

### BUGS

None.

### SEE ALSO

None.

CMOVE

NAME

CMOVE -- append Copper move instruction to user Copper list.

SYNOPSIS

CMOVE( c , a , v )

FUNCTION

Adds instruction to move value v to hardware register a.

INPUTS

c = pointer to UCopList structure  
a = hardware register  
v = 16 bit value to be written

RESULTS

This is actually a macro that calls CMove(c,&a,v) and then calls CBump(c) to bump the local pointer to the next instruction.

BUGS

ColdReset

NAME

ColdReset -- cause a system coldstart to occur

SYNOPSIS

ColdReset()

FUNCTION

This function causes a coldstart system reset sequence identical to that which occurs at power-on. All current system activities will be stopped, and the entire software system will be re-initialized. Nothing will be preserved. This function will assert processor RESET to reset all hardware devices.

EXCEPTION

This function operates in supervisor mode only. Any attempt to perform this function from user mode will result in a privilege violation trap.

## ConcatCList

### NAME

ConcatCList -- concatenate two character lists

### SYNOPSIS

```
error = ConcatCList(sourceCList, destCList)
 A0 A1
```

### FUNCTION

Exhaust the contents of the sourceCList onto the end of the destCList. The resulting destCList is the concatenation of the original destCList and sourceCList; the resulting sourceCList is empty.

### INPUTS

sourceCList -  
The cList descriptor used to manage the source character list.  
destCList -  
The cList descriptor used to manage the destination character list.

### RESULT

error -  
An error code that, if non-zero, indicates the cList pool associated with the destCList had an out-of-memory condition during the concatenation process.

## CopyCList

### NAME

CopyCList -- copy a cList to a new cList

### SYNOPSIS

```
cList = CopyCList(cList)
D0 A0
```

### FUNCTION

Copy a cList non-destructively into a new cList, created by this operation in the same cListPool.

### INPUTS

cList -  
The cList descriptor used to manage the original character list.

### RESULTS

cList -  
a longword descriptor for a cList that can be used for cList functions, and contains the same contents as the original cList.

### EXCEPTIONS

if cList is negative, not enough space was available for the new cList.

## CopySBitMap

**NAME**  
CopySBitMap -- synchronize Layer window with contents of Super BitMap

**SYNOPSIS**  
CopySBitMap( layer \* )  
a0

**FUNCTION**  
This is the inverse of SyncSBitMap.  
Copies all bits from SuperBitMap to Layer bounds.  
This is used for those functions that do not want to deal with the ClipRect structures but do want to be able to work with a SuperBitMap Layer.

**INPUTS**  
layer \* is a pointer to a Layer that has a SuperBitMap  
The Layer should already be locked by the caller.

**SEE ALSO**  
SyncSBitMap

## CreateBehindLayer

**NAME**  
CreateBehindLayer -- create a new layer behind all existing layers.

**SYNOPSIS**  
CreateBehindLayer(li,bm,x0,y0,xl,y1,flags [,bm2])  
a0 a1 d0 d1 d2 d3 d4 [ a2 ]

**INPUTS**  
li = pointer to LayerInfo structure  
bm = pointer to common BitMap used by all Layers  
bm2 = pointer to optional Super BitMap  
flags= various types of layers supported as bit sets.  
x0,y0= upper left hand corner of layer  
xl,y1= lower right hand corner of layer

**FUNCTION**  
Creates a new Layer of position and size (x0,y0)-(xl,y1)  
Makes this layer of type found in flags  
If SuperBitMap, uses bm2 as pointer to real SuperBitMap.  
and copies contents of Superbitmap into display layer.  
If this layer is a backdrop layer, places it behind all other layers, including other backdrop layers. If this is not a backdrop layer, places it behind all nonbackdrop layers.

**SEE ALSO**  
layers.h

## CreateDir

### NAME

CreateDir -- create a new directory

### SYNOPSIS

```
lock = CreateDir(name)
 DO DI
```

### FUNCTION

CreateDir creates a new directory with the name you specified, if possible. It returns an error if it fails. Remember that AmigaDOS can only create directories on devices which support them, for example, disks.

A return of zero means that AmigaDOS has found an error (such as, disk write protected), you should then call IoErr(); otherwise, CreateDir returns a shared read lock on the new directory.

### INPUTS

name - address of first character of a null-terminated string

### RESULTS

lock - BCPL pointer to a lock

## CreateExtIO

### NAME

CreateExtIO -- create an I/O request

### SYNOPSIS

```
ioReq = CreateExtIO(ioReplyPort; size);
```

### FUNCTION

Allocates memory for and initializes a new I/O request block of a user-specified number of bytes. The number of bytes MUST be greater than the length of an Exec message, or some very nasty things will happen.

### INPUTS

ioReplyPort - a pointer to an already initialized message port to be used for this I/O request's reply port.  
size - the size of the I/O request to be created.

### RESULT

Returns a pointer to the new I/O Request block, or NULL if the request failed.

### EXAMPLE

This example allocates space for IOExtTD (e.g., a trackdisk driver I/O Request block for extended I/O operations).

```
struct IORequest myBlock;
struct MsgPort port;
```

```
myBlock = CreateExtIO(port, sizeof(struct IOExtTD));
if(myBlock == NULL) {
 exit(NO_MEM_OR_SIGNALS);
}
```

### SEE ALSO

DeleteExtIO

## CreateProc

### NAME

CreateProc -- create a new process

### SYNOPSIS

```
process = CreateProc(name, pri, segment, stackSize)
D0 D1 D2 D3 D4
```

### FUNCTION

CreateProc creates a process with the name 'name'. It allocates a process control structure from the free memory area and then initializes it.

CreateProc takes a segment list as the argument 'segment'. (See also LoadSeg and UnLoadSeg.) This segment list represents the section of code that you intend to run as a new process. CreateProc enters the code at the first segment in the segment list, which should contain suitable initialization code or a jump to such.

'stackSize' represents the size of the root stack in bytes when CreateProc activates the process. 'pri' specifies the required priority of the new process. The result is the process identifier of the new process or zero if the routine failed.

The argument 'name' specifies the process name.

A zero return code implies an error of some kind.

### INPUTS

name - address of first character of a null-terminated string  
pri - integer  
segment - BCPL pointer to a segment  
stackSize - integer

### RESULTS

process - process identifier

## CreateStdIO

### NAME

CreateStdIO -- create a standard I/O request

### SYNOPSIS

```
ioStdReq = CreateStdIO(ioReplyPort)
```

### FUNCTION

Allocates memory for and initializes a new I/O request block.

### INPUTS

ioReplyPort - a pointer to an already initialized message port to be used for this I/O request's reply port.

### RESULT

Returns a pointer to the new io request block. A NULL indicates that there was not enough memory for the I/O Request, or that the reply port was not a valid port.

### EXAMPLE

```
struct IOStdReq myBlock;
struct MsgPort port;

myBlock = CreateStdIO(port);
if(myBlock == NULL) {
 printf("Insufficient memory");
}
```

### SEE ALSO

DeleteStdIO, CreateExtIO

## CreateUpfrontLayer

### NAME

CreateUpfrontLayer -- create a new layer on top of existing layers.

### SYNOPSIS

```
CreateUpfrontLayer(li,bm,x0,y0,xl,yl,flags [,bm2])
a0 a1 d0 d1 d2 d3 d4 [a2]
```

### INPUTS

li = pointer to LayerInfo structure  
bm = pointer to common BitMap used by all Layers  
bm2 = pointer to optional Super BitMap  
flags= various types of layers supported as bit sets.  
x0,y0= upper left hand corner of layer  
xl,yl= lower right hand corner of layer

### FUNCTION

Creates a new Layer of position and size (x0,y0)->(xl,yl) and places it on top of all other layers.  
Makes this layer of type found in flags.  
If SuperBitMap, uses bm2 as pointer to real SuperBitMap and copies contents of Superbitmap into display layer.

### SEE ALSO

layers.h

## CurrentDir

### NAME

CurrentDir -- make a directory associated with a lock the current working directory

### SYNOPSIS

```
oldLock = CurrentDir(lock)
D0 D1
```

### FUNCTION

CurrentDir makes current a directory associated with a lock. (See also LOCK). It returns the old current directory lock.

A value of zero is a valid result here and indicates that the current directory is the root of the initial start-up disk.

### INPUTS

lock - BCPL pointer to a lock

### RESULTS

oldLock - BCPL pointer to a lock



## CurrentTime

### NAME

CurrentTime -- get the current time values

### SYNOPSIS

```
ULONG Seconds, Micros;
CurrentTime(&Seconds, &Micros)
 D0 D1
```

### FUNCTION

This function puts copies of the current time into the supplied argument pointers. This time value is not extremely accurate, nor is it of a very fine resolution. The time will be updated no more than sixty times a second and will typically be updated far fewer times a second.

### INPUTS

Seconds = pointer to a ULONG variable to receive the current seconds value.  
Micros = pointer to a ULONG variable for the current microseconds value.

### RESULT

Puts the time values into the memory locations specified by the arguments.

### BUGS

None.

### SEE ALSO

None.

## CWAIT

### NAME

CWAIT -- append Copper wait instruction to user Copper list.

### SYNOPSIS

```
CWAIT(c , v , h)
```

### FUNCTION

Adds instruction to wait for vertical beam position v and horizontal position h

### INPUTS

c = pointer to UCopList structure  
v = vertical beam position (relative to top of ViewPort)  
h = horizontal beam position

### RESULTS

This is actually a macro that calls CWait(c,v,h) and then calls CBump(c) to bump the local pointer to the next instruction.

### BUGS

## DateStamp

### NAME

DateStamp -- obtain the date and time in internal forma

### SYNOPSIS

DateStamp( v );

### FUNCTION

DateStamp takes a vector of three longwords that is set to the current time. The first element in the vector is a count of the number of days. The second element is the number of minutes elapsed in the day. The third is the number of ticks elapsed in the current minute. A tick happens 50 times a second. DateStamp ensures that the day and minute are consistent. All three elements are zero if the date is unset. DateStamp currently only returns even multiples of 50 ticks. Therefore the time you get is always an even number of ticks.

### INPUTS

v - address of the first element in an array of three longwords

### RESULTS

This array is filled as described under FUNCTION.

## Deallocate

### NAME

Deallocate -- deallocate a block of memory

### SYNOPSIS

Deallocate( freeList, memoryBlock, byteSize)  
                  A0                  A1                  D0

### FUNCTION

This function deallocates memory by returning it to the appropriate free memory pool. This function can be used to free an entire block allocated with the above function, or it can be used to free a sub-block of a previously allocated block.

If memoryBlock is not on a block boundary (MEM\_BLOCKSIZE) then it will be rounded down. Note that this will work correctly with all the memory allocation routines, but may cause surprises if one is freeing only part of a region. If byteSize is null, nothing happens. Also, the size of the block will be rounded up, so the freed block will fill an entire memory block.

### INPUTS

freeList - points to the free list  
memoryBlock - memory block to return  
byteSize - the size of the desired block in bytes

### SEE ALSO

Allocate

## Delay

### NAME

Delay -- delay a process for a specified time

### SYNOPSIS

```
Delay(timeout)
 Dl
```

### FUNCTION

The function Delay takes an argument 'timeout'. 'timeout' allows you to specify how long the process should wait in ticks (50 per second).

### INPUTS

timeout - integer

## DeleteExtIO

### NAME

DeleteExtIO -- return memory allocated for extended I/O request

### SYNOPSIS

```
DeleteExtIO(ioReq);
```

### FUNCTION

Frees up an IO request as allocated by CreateExtIO().

### INPUTS

ioReq - A pointer to the IORequest block to be freed.

### RESULTS

No return value

### EXAMPLE

```
struct IORequest ioReq;
DeleteExtIO(ioReq);
```

### SEE ALSO

CreateExtIO

## DeleteFile

### NAME

DeleteFile -- delete a file or directory

### SYNOPSIS

```
success = DeleteFile(name)
 D0 D1
```

### FUNCTION

DeleteFile attempts to delete the file or directory 'name'. It returns an error if the deletion fails. Note that you must delete all the files within a directory before you can delete the directory itself.

### INPUTS

name - address of first character of a null-terminated string

### RESULTS

success - boolean

## DeleteLayer

### NAME

DeleteLayer -- delete layer from layer list.

### SYNOPSIS

```
DeleteLayer(li, l)
 a0 a1
```

### INPUTS

li = pointer to LayerInfo structure  
l = pointer to a layer

### FUNCTION

Removes this layer from the list of layers and releases memory associated with it. Restores other layers that may have been obscured by it. Triggers refresh in those that may need it. If this is a superbitmap, makes sure SuperBitMap is current. The SuperBitMap is not removed from the system but is available for program without rest of layer stuff.

### SEE ALSO

layers.h

## DeviceProc

### NAME

DeviceProc -- return the process identifier of the process handling that I/O

### SYNOPSIS

```
process = DeviceProc(name)
D0 DI
```

### FUNCTION

DeviceProc returns the process identifier of the process that handles the device associated with the specified name. If DeviceProc cannot find a process handler, the result is zero. If 'name' refers to a file on a mounted device, then IoErr() returns a pointer to a directory lock.

You can use this function to determine the process identification of the handler process where the system should send its messages.

### INPUTS

name - address of first character of a null-terminated string

### RESULTS

process - BCPL pointer to a Process

## Disable

### NAME

Disable -- Disable interrupts in a non-preemptive fashion.

### SYNOPSIS

```
Disable();
```

### FUNCTION

Disabling is similar to forbidding, but it also prevents interrupts from occurring during a critical section. Disabling is required when a task accesses structures that are shared by interrupt code. It eliminates the possibility of an interrupt accessing shared structures by preventing interrupts from occurring.

To disable interrupts you can call the Disable() function. If you are writing in assembly code, the DISABLE macro is more efficient (but consumes more code space). To enable interrupts again, use the Enable() function and ENABLE macros.

Like forbidden sections, disabled sections can be nested. Also like forbidden sections, the Wait() function implies an Enable() until the task again regains the processor.

It is important to realize that there is a danger in using disabled sections. Because the software on the Amiga depends heavily on its interrupts occurring in nearly real time, you cannot disable for more than a very brief instant. A rule of thumb is to disable for no more than 250 microseconds.

Masking interrupts by changing the 68000 processor interrupt priority levels with the MOVESR instruction can also be dangerous and is generally discouraged. The disable- and enable-related functions and macros control interrupts through the 4703 custom chip and not through the 68000 priority level. In addition, the processor priority level can be altered only from supervisor mode (which means this process is much less efficient).

It is never necessary to both disable and forbid. Because disable prevents interrupts, it also prevents preemptory task scheduling. Many Exec lists can only be accessed while disabled. Suppose you want to print the names of all waiting tasks. You would need to access the task list from a disabled section. In addition, you must avoid calling certain system functions that require multitasking to function properly (printf() for example). In this example, the names are gathered into a name array while the code section is disabled. Then the code section is enabled and the names are printed.

```
struct ExecBase *eb;
struct Task *tc;
char *names[ARRAYSIZE];
int count;

Disable();
for (tc = eb -> TaskWait.tc_Node.lh_Head;
```

```

 tc -> tc_Node.In_Succ;
 tc = tc -> tc_Node.In_Succ) {
 names[count++] = tc -> tc_Node.In_Name;
}
Enable();
for (i = 0; i < count; i++) {
 printf (" %s ", names[i]);
}

```

Of course, the code in this example will have problems if a waiting task is removed before its name is printed. If this were to happen, the name-string pointer would no longer be valid. To avoid such problems it is a good programming practice to copy the entire name string into a temporary buffer.

DisownBlitter

NAME

DisownBlitter -- return blitter to free state.

SYNOPSIS

DisownBlitter()

FUNCTION

Free blitter for use by other blitter users

INPUTS

RETURNS

SEE ALSO

OwnBlitter

## DisplayAlert

### NAME

DisplayAlert -- create a display of an alert message

### SYNOPSIS

```
DisplayAlert(AlertNumber, String, Height)
 D0 A0 D1
```

### FUNCTION

Creates an alert display with the specified message.

If the system can recover from this alert, it is a RECOVERY\_ALERT. The routine waits until the user presses one of the mouse buttons, after which the display is restored to its original state and a BOOL value is returned by this routine to specify whether or not the user pressed the left mouse button.

If the system cannot recover from this alert, it is a DEADEND\_ALERT, and this routine returns immediately upon creating the alert display. The return value is FALSE.

The AlertNumber is a LONG value, related to the value sent to the Alert() routine. The only bits that are pertinent to this routine, however, are the ALERT\_TYPE bits. These bits must be set to RECOVERY\_ALERT for alerts from which the system may safely recover or DEADEND\_ALERT for fatal alerts. These states are described in the paragraph above. A third type of alert, the DAISY\_ALERT, is used only by the Executive.

The String argument points to an AlertMessage string. The AlertMessage string is composed of one or more substrings, each of which contains the following components:

- o First, a 16-bit x coordinate and an 8-bit y coordinate, describing where on the alert display you want this string to appear. The y coordinate describes the offset to the baseline of the text.
- o Then, the bytes of the string itself, which must be null-terminated (end with a byte of zero).
- o Lastly, the continuation byte, which specifies whether or not another substring follows this one. If the continuation byte is non-zero, there is another substring to be processed in this AlertMessage. If the continuation byte is zero, this is the last substring in the message.

The last argument, Height, describes how many video lines tall you want the alert display to be.

### INPUTS

AlertNumber = the number of this AlertMessage. The only pertinent bits of this number are the ALERT\_TYPE bits. The rest of the number is ignored by this routine.  
String = pointer to the alert message string, as described above.  
Height = minimum display lines required for your message.

### RESULT

A BOOL value of TRUE or FALSE. If this is a DEADEND\_ALERT, FALSE is always the return value. If this is a RECOVERY\_ALERT, the return value will be TRUE if the user presses the left mouse button in response to your message and FALSE if the user presses the right button.

### BUGS

If the system is in more trouble than you think, the level of your alert may become DEADEND\_ALERT without you ever knowing about it.

### SEE ALSO

None

## DisplayBeep

### NAME

DisplayBeep -- "beep" the video display

### SYNOPSIS

```
DisplayBeep(Screen)
 A0
```

### FUNCTION

"Beeps" the video display by flashing the background color of the specified screen. If the Screen argument is NULL, every screen in the display will be beeped. Flashing all screens is not a polite thing to do, so this should be reserved for dire circumstances.

Such a routine is supported because the Amiga has no internal bell or speaker. When the user needs to know of an event that is not serious enough to require the use of a requester, the DisplayBeep() function should be called.

### INPUTS

Screen = pointer to a Screen structure. If NULL, every Intuition screen will be flashed.

### RESULT

None

### BUGS

None

### SEE ALSO

None

## DisposeLayerInfo

### NAME

DisposeLayerInfo -- return all memory for LayerInfo to mem pool

### SYNOPSIS

```
DisposeLayerInfo(li)
 a0
```

### INPUTS

li = pointer to LayerInfo structure

### FUNCTION

Returns LayerInfo and any other memory attached to this LayerInfo to memory allocator

### SEE ALSO

layers.h



## DisposeRegion

### NAME

DisposeRegion -- return all space for this region to free  
memory pool

### SYNOPSIS

```
DisposeRegion(region)
 a0
```

### Function

Frees all RegionRectangles for this Region and then  
frees the Region itself

### INPUTS

region = pointer to Region structure

### BUGS

## DoCollision

### NAME

DoCollision -- tests every GEL in GEL list for collisions

### SYNOPSIS

```
DoCollision(RPort)
 a1
```

### FUNCTION

Tests each GEL in GEL list for boundary and GEL-to-GEL collisions  
On detecting one of these collisions, the appropriate collision-handling  
routine is called. See the documentation for a thorough description of  
which collision routine is called.

This routine expects to find the GEL list correctly sorted in Y,X order.  
The system routine SortGList performs this function for the user

### INPUTS

RPort = pointer to a struct RastPort

### RESULT

Nothing

### BUGS

Does not handle GEL-to-GEL collisions completely correctly

### SEE ALSO

SortGList

## DoIO

### NAME

DoIO -- perform an I/O command and wait for completion

### SYNOPSIS

```
error = DoIO(iORequest)
D0 A1
```

### FUNCTION

This function requests a device driver to perform the I/O command specified in the I/O request. This function will always block until the I/O request is completed.

### INPUTS

iORequest - pointer to a properly initialized I/O request

### RESULTS

error - see WaitIO

### SEE ALSO

SendIO, WaitIO

## DoubleClick

### NAME

DoubleClick -- test two time values for double-click timing

### SYNOPSIS

```
DoubleClick(StartSeconds, StartMicros, CurrentSeconds, CurrentMicros)
D0 D1 D2 D3
```

### FUNCTION

Compares the difference in the time values with the double-click timeout range that the user (using the Preferences tool or some other source) has configured into the system. If the difference between the specified time values is within the current double-click time range, this function returns TRUE; otherwise, it returns FALSE.

These time values can be found in InputEvents and IDCMP messages. The time values are not perfect; however, they are precise enough for nearly all applications.

### INPUTS

StartSeconds, StartMicros = the timestamp value describing the start of the double-click time period you are considering.

CurrentSeconds, CurrentMicros = the timestamp value describing the end of the double-click time period you are considering.

### RESULT

If the difference between the supplied timestamp values is within the double-click time range in the current set of Preferences, this function returns TRUE; otherwise, it returns FALSE.

### BUGS

None

### SEE ALSO

CurrentTime()

Draw

NAME

Draw -- draw a line between the current pen position  
and the new x,y position

SYNOPSIS

Draw( rp, x, y)  
      A1 D0 D1

FUNCTION

Draws a line from the current pen position to (x,y).

INPUTS

rp       pointer to a RastPort  
x,y      point in the RastPort to end the line.

DrawBorder

NAME

DrawBorder -- draw the specified border into the RastPort

SYNOPSIS

DrawBorder(RastPort, Border, LeftOffset, TopOffset)  
                  A0            A1            D0            D1

FUNCTION

First, this function sets up the drawing mode and pens in the RastPort according to the arguments of the Border structure. Then, it draws the vectors of the Border argument into the RastPort, offset by the LeftOffset and TopOffset. This routine does Intuition window clipping as appropriate; if you draw a line outside of your window, your imagery will be clipped at the window's edge.

If the NextBorder field of the Border argument is non-zero, the next Border is rendered as well (return to the top of this FUNCTION section for details).

INPUTS

RastPort = pointer to the RastPort to receive the border crossing.  
Border = pointer to a Border structure.  
LeftOffset = the offset that will be added to each vector's x coordinate.  
TopOffset = the offset that will be added to each vector's y coordinate.

RESULT

None

BUGS

None

SEE ALSO

None

## DrawGList

### NAME

DrawGList -- process the GEL list, queueing VSprites, drawing Bobs

### SYNOPSIS

DrawGList(RPort, VPort)        as called by C  
      al        a0

### FUNCTION

Performs one pass of the current GEL list

- If nextLine and lastColor are defined, these are initialized for each GEL.
- If it's a VSprite, build it into the Copper list
- If it's a Bob, draw it into the current raster
- Copy the save values into the "old" variables, double-buffering if required

### INPUTS

a1 = pointer to the RastPort where Bobs will be drawn  
a5 = pointer to GfxBase

### RESULT

Nothing

### BUGS

MUSTDRAW is not implemented yet and probably will not be for this release. We are sad.

### SEE ALSO

Nothing

## DrawImage

### NAME

DrawImage -- draw the specified Image into the RastPort

### SYNOPSIS

DrawImage(RastPort, Image, LeftOffset, TopOffset)  
          A0        A1        D0        D1

### FUNCTION

First, this function sets up the drawing mode and pens in the RastPort according to the arguments of the Image structure. Then, it moves the image data of the Image argument into the RastPort, offset by the LeftOffset and TopOffset. This routine does Intuition window clipping as appropriate; if you draw an image outside of your window, your imagery will be clipped at the window's edge.

If the NextImage field of the Image argument is non-zero, the next Image is rendered as well (return to the top of this section for details).

### INPUTS

RastPort = pointer to the RastPort to receive the border crossing.  
Image = pointer to an Image structure.  
LeftOffset = the offset that will be added to the Image's x coordinate.  
TopOffset = the offset that will be added to the Image's y coordinate.

### RESULT

None

### BUGS

None

### SEE ALSO

None

## DupLock

### NAME

DupLock -- duplicate a lock

### SYNOPSIS

```
newLock = DupLock(lock)
 D0 D1
```

### FUNCTION

DupLock takes a shared filing system read lock and returns another shared read lock to the same object. It is impossible to create a copy of a write lock. (For more information on locks, see under LOCK.)

### INPUTS

lock - BCPL pointer to a lock

### RESULTS

newLock - BCPL pointer to a lock

## Enable

### NAME

Enable -- Enable interrupts following a Disable()

### SYNOPSIS

```
Enable();
```

### FUNCTION

Interrupts will not necessarily be enabled after this call since the Disable() function nests (only an equal number of Enable's following a set of Disable's finally re-enables interrupts).

### SEE ALSO

Disable

EndRefresh

NAME

EndRefresh -- end the optimized refresh state of the window

SYNOPSIS

EndRefresh(Window, Complete)  
A0 D0

FUNCTION

This function gets you out of the special refresh state of your window. It is called following a call to BeginRefresh(), which begins the special refresh state. While your window is in the refresh state, the only drawing that will be wrought in your window will be to those areas that were recently revealed and that need to be refreshed.

After your program has done all the needed refreshing for this window, this routine is called to restore the window to its non-refreshing state. Then all rendering will go to the entire window as usual.

The Complete argument is a Boolean TRUE or FALSE value used to describe whether or not the refreshing that has been done is all that needs to be done at this time. Most often, this argument will be TRUE. However, if, for instance, you have multiple tasks or multiple procedure calls that must run to completely refresh the window, each can call its own Begin/EndRefresh() pair with a Complete argument of FALSE, and only the last calls with a Complete argument of TRUE.

INPUTS

Window = pointer to the Window currently in optimized-refresh mode.  
Complete = Boolean TRUE or FALSE describing whether or not this window is completely refreshed.

RESULT

None

BUGS

None

SEE ALSO

BeginRefresh()

EndRequest

NAME

EndRequest -- end the request and reset the window

SYNOPSIS

EndRequest(Requester, Window)  
A0 A1

FUNCTION

This function ends the request by erasing the requester and resetting the window. Note that this does not necessarily clear all requesters from the window, only the specified one. If the window labors under other requesters, they will remain in the window.

INPUTS

Requester = pointer to the structure of the requester to be removed.  
Window = pointer to the Window structure with which this requester is associated.

RESULT

None

BUGS

None

SEE ALSO

None

## EndUpdate

### NAME

EndUpdate -- remove damage list and restore state of layer to normal.

### SYNOPSIS

```
EndUpdate(l, flag)
 a0 d0
```

### INPUTS

l = pointer to a layer  
flag= TRUE if update was successful. The damage list is cleared.

### FUNCTION

After the programmer has redrawn his picture, he calls this routine to restore the ClipRects to point to his standard layer tiling. Use flag=0 if you are only making a partial update. You may use the other region functions to clip adjust the DamageList to reflect a partial update.

### SEE ALSO

layers.h BeginUpdate()

## Enqueue

### NAME

Enqueue -- insert or append node to a system queue

### SYNOPSIS

```
Enqueue(list, node)
 A0 A1
```

### FUNCTION

Insert or append a node into a system queue. The insert is performed based on the node priority -- it will keep the list properly sorted. New nodes will be inserted in front of the first node with a lower priority. Hence a FIFO queue for nodes of equal priority

### INPUTS

list - a pointer to the system queue header  
node - the node to enqueue

## Examine

### NAME

Examine -- examine a directory or file associated with a lock

### SYNOPSIS

```
success = Examine(lock, FileInfoBlock)
 D0 D1 D2
```

### FUNCTION

Examine fills in information in the FileInfoBlock concerning the file or directory associated with the lock. This information includes the name, size, creation date, and whether it is a file or directory.

Note: FileInfoBlock must be longword-aligned. You can ensure this in C if you use Allocmem. (See the "Amiga ROM Kernel Reference Manual: Exec" for further details on the exec call Allocmem.)

Examine gives a return code of zero if it fails.

### INPUTS

lock - BCPL pointer to a lock  
FileInfoBlock - address of a file info block

### RESULTS

success - boolean

## Execute

### NAME

Execute -- execute a CLI command

### SYNOPSIS

```
Success = Execute(commandString, input, output)
 D0 D1 D2 D3
```

### FUNCTION

This function takes a string (commandString) that specifies a CLI command and arguments, and attempts to execute it. The CLI string can contain any valid input that you could type directly at a CLI, including input and output indirection.

The input file handle will normally be zero, and in this case the EXECUTE command will perform whatever was requested in the commandString and then return. If the input file handle is nonzero, after the (possibly null) commandString is performed, subsequent input is read from the specified input file handle until end-of-file is reached.

In most cases, the output file handle must be provided and will be used by the CLI commands as their output stream unless redirection was specified. If the output file handle is set to zero, the current window, normally specified as \*, is used. Note that programs running under the Workbench do not normally have a current window.

The Execute function may also be used to create a new interactive CLI process just like those created with the NEWCLI function. To do this, you should call Execute with an empty commandString, and pass a file handle relating to a new window as the input file handle. The output file handle should be set to zero. The CLI will read commands from the new window, and will use the same window for output. This new CLI window can only be terminated by using the ENDCLI command. For this command to work the program C:RUN must be present in C:.

### INPUTS

commandString - address of first character of a null-terminated string  
input - BCPL pointer to a file handle  
output - BCPL pointer to a file handle

### RESULTS

Success - boolean



Exit

NAME

Exit -- exit from a program

SYNOPSIS

```
Exit(returnCode)
 D1
```

FUNCTION

Exit acts differently depending on whether you are running a program under a CLI or not. If you run a program that calls Exit as a command under a CLI, the command finishes and control reverts to the CLI. Exit then interprets the argument 'returnCode' as the return code from the program.

If you run the program as a distinct process, Exit deletes the process and releases the space associated with the stack, segment list, and process structure.

INPUTS

returnCode - integer

ExNext

NAME

ExNext -- examine the next entry in a directory.

SYNOPSIS

```
success = ExNext(lock, FileInfoBlock)
 D0 D1 D2
```

FUNCTION

This routine is passed a lock, usually associated with a directory, and a FileInfoBlock filled in by a previous call to Examine. The FileInfoBlock contains information concerning the first file or directory stored in the directory associated with the lock. ExNext also modifies the FileInfoBlock so that subsequent calls return information about each following entry in the directory.

ExNext gives a return code of zero if it fails for some reason. One reason for failure is reaching the last entry in the directory. However, IoErr() holds a code that may give more information on the exact cause of a failure. When ExNext finishes after the last entry, it returns ERROR\_NO\_MORE\_ENTRIES

Follow these steps to examine a directory:

- 1) Use Examine to get a FileInfoBlock about the directory you wish to examine.
- 2) Pass ExNext the lock related to the directory and the FileInfoBlock filled in by the previous call to Examine.
- 3) Keep calling ExNext until it fails with the error code held in IoErr() equal to ERROR\_NO\_MORE\_ENTRIES.
- 4) Note that if you don't know what you are examining, inspect the type field of the FileInfoBlock returned from Examine to find out whether it is a file or a directory which is worth calling ExNext for.

The type field in the FileInfoBlock has two values: if it is negative, then the file system object is a file; if it is positive, then it is a directory.

INPUTS

lock - BCPL pointer to a lock  
FileInfoBlock - pointer to a file info block

RESULTS

success - boolean

SPECIAL NOTE

The FileInfoBlock must be longword-aligned.

faddi

NAME

faddi -- add two floating-point numbers

C USAGE

```
fnum3 = fnum1 + fnum2;
 D1 D0
```

FUNCTION

Accepts two floating-point numbers and returns the arithmetic sum of said numbers. Note that this function is called by compiler-generated code, not by a user-generated function call.

INPUTS

fnum1 - floating-point number  
fnum2 - floating-point number

RESULT

fnum3 - floating-point number

BUGS

None

SEE ALSO

SPAdd,

FattenLayerInfo

NAME

FattenLayerInfo -- convert 1.0 LayerInfo to 1.1 LayerInfo

SYNOPSIS

```
FattenLayerInfo(li)
 a0
```

INPUTS

li = pointer to LayerInfo structure

FUNCTION

From 1.1 software and on, need to have more info in the Layer\_Info structure. To do this in a 1.0-supportable manner requires allocation and deallocation of the memory whenever most layer library functions are called. To prevent unnecessary allocation/deallocation, FattenLayerInfo will preallocate the necessary data structures and fool the layer library into thinking it has a LayerInfo gotten from NewLayerInfo. NewLayerInfo is the approved method for getting this structure. When a program needs to give up the LayerInfo structure, it must call ThinLayerInfo before freeing the memory. ThinLayerInfo is not necessary if New/DisposeLayerInfo are used, however.

SEE ALSO

NewLayerInfo ThinLayerInfo DisposeLayerInfo layers.h

fcmpi

NAME

fcmpi -- compare two floating-point numbers and set appropriate condition codes

C USAGE

```
if (fnum1 <= fnum2) {...}
 D1 D0
```

FUNCTION

Accepts two floating-point numbers and returns the condition codes set to indicate the result of said comparison. Note that this function is called by compiler-generated code, not by a user-generated function call.

INPUTS

fnum1 - floating-point number  
fnum2 - floating-point number

RESULT

Condition codes set to reflect the following branches:

GT - fnum2 > fnum1  
GE - fnum2 >= fnum1  
EQ - fnum2 = fnum1  
NE - fnum2 != fnum1  
LT - fnum2 < fnum1  
LE - fnum2 <= fnum1

BUGS

None

SEE ALSO

SPCmp,

fdivi

NAME

fdivi -- divide two floating-point numbers

C USAGE

```
fnum3 = fnum1 / fnum2;
 D1 D0
```

FUNCTION

Accepts two floating-point numbers and returns the arithmetic division of said numbers. Note that this function is called by compiler-generated code, not by a user-generated function call.

INPUTS

fnum1 - floating-point number  
fnum2 - floating-point number

RESULT

fnum3 - floating-point number

BUGS

None

SEE ALSO

SPDiv,

fflti

NAME

fflti -- convert integer number to fast floating point

C USAGE

```
fnum = (FLOAT) inum;
 DO
```

FUNCTION

Accepts an integer and returns the converted floating-point result of said number. Note that this function is called by compiler-generated code, not by a user-generated function call.

INPUTS

inum - signed integer number

RESULT

fnum - floating-point number

BUGS

None

SEE ALSO

SPFlt,

FindName

NAME

FindName -- find a system list node with a given name

SYNOPSIS

```
node = FindName(start, name)
 DO A0 A1
```

FUNCTION

Traverse a system list until a node with the given name is found. To find multiple occurrences of a string, this function may be called with a node starting point.

INPUTS

start - a list header or a list node to start the search (if node, this one is skipped)  
name - a pointer to a name string terminated with null

RESULTS

node - a pointer to the node with the same name else zero to indicate that the string was not found.

## FindPort

### NAME

FindPort -- find a given system message port

### SYNOPSIS

```
port = FindPort(name)
D0 A1
```

### FUNCTION

This function will search the system message port list for a port with the given name. The first port matching this name will be returned.

### INPUT

name - name of the port to find

### RETURN

port - a pointer to the message port, or zero if not found.

## FindTask

### NAME

FindTask -- find a task with the given name or find oneself

### SYNOPSIS

```
task = FindTask(name)
D0 A1
```

### FUNCTION

This function will check all task queues for a task with the given name, and return a pointer to its task control block. If a null name pointer is given a pointer to the current task will be returned.

### INPUT

name - pointer to a name string

### RESULT

task - pointer to the task

## FindToolType

NAME  
FindToolType -- find the value of a ToolType variable

SYNOPSIS  
value = FindToolType( toolTypeArray, typeName )  
D0                    A0                    A1

FUNCTION  
This function searches a tool type array for a given entry and returns a pointer to that entry. This is useful for finding standard tool type variables. The returned value is not a new copy of the string but is only a pointer to the part of the string after typeName.

INPUTS  
toolTypeArray - an array of strings  
typeName - the name of the tooltype entry

RESULTS  
value - a pointer to a string that is the value bound to typeName, or NULL if typeName is not in the toolTypeArray.

EXCEPTIONS

EXAMPLE  
Assume the tool type array has two strings in it:  
"FILETYPE=text"  
"TEMPDIR=:t"

FindToolType( toolTypeArray, "FILENAME" ) returns "text"  
FindToolType( toolTypeArray, "TEMPDIR" ) returns ":t"  
FindToolType( toolTypeArray, "MAXSIZE" ) returns NULL

SEE ALSO  
MatchToolValue

BUGS

## Flood

NAME  
Flood -- flood rastport like areafill

SYNOPSIS  
Flood( rp, mode, x, y )  
          a1 d2 d0 d1

FUNCTION  
Searches the BitMap starting at (x,y). Fills all adjacent pixels if they:  
a: are not the same as AOLPen                    Mode 0  
a: are the same as the one at (x,y)            Mode 1

When actually doing the fill, uses the modes that apply to standard area-fill routines such as drawmodes and patterns.

INPUTS  
rp            pointer to RastPort  
(x,y)        coordinate in BitMap  
mode        0    fill all adjacent pixels searching for border  
              1    fill all adjacent pixels that have same pen number as (x,y)

SEE ALSO

BUGS

None known

## FlushCList

### NAME

FlushCList -- clear a character list

### SYNOPSIS

```
FlushCList(cList)
 AO
```

### FUNCTION

ensure that the cList is empty.

### INPUTS

cList -  
The cList header used to manage this character list,  
as returned by AllocCList or StrToCL.

### RESULTS

## fmul

### NAME

fmul -- multiply two floating-point numbers

### C USAGE

```
fnum3 = fnum1 * fnum2;
 D1 D0
```

### FUNCTION

Accepts two floating-point numbers and returns the arithmetic multiplication of said numbers. Note that this function is called by compiler-generated code, not by a user-generated function call.

### INPUTS

fnum1 - floating-point number  
fnum2 - floating-point number

### RESULT

fnum3 - floating-point number

### BUGS

None

### SEE ALSO

SPMul,

fnegi

NAME

fnegi -- negate the supplied floating-point number

C USAGE

```
fnum2 = -fnum1;
 DO
```

FUNCTION

Accepts a floating-point number and returns the value of said number after having been subtracted from 0.0. Note that this function is called by compiler-generated code, not by a user-generated function call.

INPUTS

fnum1 - floating-point number

RESULT

fnum2 - floating-point negation of fnum1

BUGS

None

SEE ALSO

SPNeg,

Forbid

NAME

Forbid -- prevent task rescheduling on a non-preemptive basis.

SYNTAX

```
Forbid();
```

FUNCTION

Forbidding is used when a task is accessing shared structures that might also be accessed at the same time from another task. It effectively eliminates the possibility of simultaneous access by imposing nonpreemptive task scheduling. This has the net effect of disabling multitasking for as long as your task remains in its running state. While forbidden, your task will continue running until it performs a call to Wait() or exits from the forbidden state. Interrupts will occur normally, but no new tasks will be dispatched, regardless of their priorities.

When a task running in the forbidden state calls the Wait() function, it implies a temporary exit from its forbidden state. While the task is waiting, the system will perform normally. When the task receives one of the signals it is waiting for, it will again reenter the forbidden state. To become forbidden, a task calls the Forbid() function. To escape, the Permit() function is used. The use of these functions may be nested with the expected affects; you will not exit the forbidden mode until you call the outermost Permit().

As an example, Exec memory region lists should be accessed only when forbidden. To access these lists without forbidding jeopardizes the integrity of the entire system.

```
struct ExecBase *eb;
struct MemHeader *mh;
APTR firsts[ARRAYSIZE];
int count;

Forbid();
for (mh = (struct MemHeader *) eb -> MemList.lh_Head;
 mh -> mh_Node.ln_Succ;
 mh = mh -> mh_Node.ln_Succ) {
 firsts[count++] = mh -> mh_First;
}
Permit();
```

As this program traverses down the memory region list, it remains forbidden to prevent the list from changing as it is being accessed.



## FreeCList

### NAME

FreeCList -- free a cList

### SYNOPSIS

```
FreeCList(cList)
 A0
```

### FUNCTION

Release the cList descriptor and any resources it uses.  
References to the cList are no longer valid.

### INPUTS

cList -  
a descriptor for a cList that is no longer to be used.

### NOTES

This function is implicitly performed by CLToBuf.

## FreeColorMap

### NAME

FreeColorMap -- free the ColorMap structure and return memory  
to free memory pool

### SYNOPSIS

```
FreeColorMap(colormap)
 a0
```

### INPUTS

colormap pointer to ColorMap allocated with GetColorMap

### RESULT

The space is made available for others to use.

### BUGS

### SEE ALSO

SetRGB4 GetColorMap

## FreeCopList

NAME FreeCopList -- deallocate intermediate Copper list

SYNOPSIS FreeCopList(coplist)

FUNCTION Deallocates all memory associated with this Copper list

INPUTS coplist = pointer to structure CopList

RESULTS memory returned to memory manager

BUGS none known

SEE ALSO

## FreeCprList

NAME FreeCprList -- deallocate hardware Copper list

SYNOPSIS FreeCprList(cprlist)

FUNCTION Return cprlist to free memory pool

INPUTS cprlist = pointer to cprlist structure

RESULTS

BUGS none known

SEE ALSO

## FreeDiskObject

### NAME

FreeDiskObject -- free all memory in a Workbench disk object

### SYNOPSIS

```
FreeDiskObject(diskobj)
 A0
```

### FUNCTION

This routine frees all memory in a Workbench disk object and also frees and the object itself. It is implemented via FreeFreeList().

GetDiskObject() takes care of all the initialization required to set up the objects free list. This procedure may ONLY be called on DiskObject allocated via GetDiskObject().

### INPUTS

diskobj -- a pointer to a DiskObject structure

### RESULTS

### EXCEPTIONS

### SEE ALSO

GetDiskObject, FreeFreeList

### BUGS

## FreeEntry

### NAME

FreeEntry -- free many regions of memory

### SYNOPSIS

```
FreeEntry(memList)
 A0
```

### FUNCTION

This routine takes a memList structure (as returned by AllocEntry) and frees all the entries.

### INPUTS

memList -- pointer to structure filled in with memEntry structures

## FreeFreeList

### NAME

FreeFreeList -- free all memory in a free list

### SYNOPSIS

```
FreeFreeList(free)
 A0
```

### FUNCTION

This routine frees all memory in a free list, and the free list itself. It is useful for easily getting rid of all memory in a series of structures. There is a free list in a Workbench object, and this contains all the memory associated with that object.

A FreeList is a list of MemList structures. See the MemList and MemEntry documentation for more information.

If the FreeList itself is in the free list, it must be in the first MemList in the FreeList.

### INPUTS

free -- a pointer to a FreeList structure

### RESULTS

### EXCEPTIONS

### SEE ALSO

AllocEntry, FreeEntry, AddFreeList

### BUGS

## FreeGBuffers

### NAME

FreeGBuffers -- deallocate memory gotten by GetGBuffers

### SYNOPSIS

```
FreeGBuffers(anOb, RPort, db) as called by C
 a0 al d0
```

### FUNCTION

For each sequence of each component of the AnimOb, deallocates memory for:

SaveBuffer

BorderLine

CollMask and ImageShadow (point to same buffer)

if db is set (user wants double-buffering) deallocate:

DBufPacket

BufBuffer

### INPUTS

a1 = pointer to the AnimOb structure

a2 = pointer to the current RastPort

d0 = double-buffer indicator (set TRUE for double-buffering)

### RESULT

### BUGS

None known

### SEE ALSO

Nothing

## FreeMem

### NAME

FreeMem -- deallocate with knowledge

### SYNOPSIS

```
FreeMem(memoryBlock, byteSize)
 AI DO
```

### FUNCTION

Free a region of memory, returning it to the pool from which it came.

### INPUTS

memoryBlock - memory block to free  
If the memoryBlock previously returned by an allocation routine.  
byteSize - the size of the block in bytes

### SEE ALSO

AllocMem, AllocAbs

## FreeRaster

### NAME

FreeRaster -- release an allocated area to the system free memory pool.

### SYNOPSIS

```
FreeRaster(p, width, height)
 a0 d0 d1
```

### INPUTS

p = a pointer to a memory space returned as a result of a call to AllocRaster

width = the width in bits of the bitplane

height = the height in bits of the bitplane

the same values of width and height with which you called AllocRaster in the first place, when the pointer p returned. This defines the size of the memory space which is to be returned to the free memory pool.

### FUNCTION

Returns to the free memory pool the memory space that had been allocated by a call to AllocRast.

NOTE: Always use the same values that were used with AllocRaster

## FreeRemember

### NAME

FreeRemember -- free the memory allocated by calls to AllocRemember()

### SYNOPSIS

```
FreeRemember(RememberKey, ReallyForget)
 A0 D0
```

### FUNCTION

This function frees up memory allocated by the AllocRemember() function. It will free up just the Remember structures, which supply the link nodes that tie your allocations together, or it will deallocate both the link nodes and your memory buffers.

If you want to deallocate just the Remember structure link nodes, you should set the ReallyForget argument to FALSE. However, if you want FreeRemember() to really forget about all the memory, including both the Remember structure link nodes and the buffers you requested via earlier calls to AllocRemember(), you should set the ReallyForget argument to TRUE.

If you're not sure whether or not you want to Really Forget, refer to figure 11-1.

### INPUTS

RememberKey = the address of a pointer to a Remember structure. This pointer should either be NULL or be set to some value (possibly NULL) by a call to AllocRemember(). For example:

```
struct Remember *RememberKey;
RememberKey = NULL;
AllocRemember(&RememberKey, BUFSIZE, MEMF_CHIP)
FreeRemember(&RememberKey, TRUE)
```

ReallyForget = a BOOL FALSE or TRUE describing, respectively, whether you want to free up only the Remember nodes or whether you want this procedure to really forget about all of the memory, including both the nodes and the memory buffers pointed to by the nodes.

### RESULT

None

### BUGS

None

### SEE ALSO

AllocRemember()

## FreeSignal

### NAME

FreeSignal -- free a signal bit

### SYNOPSIS

```
FreeSignal(signalNum)
 D0
```

### FUNCTION

This function frees a previously allocated signal bit for reuse. This call must be performed while running in the same task in which the signal was allocated.

### WARNING

Signals may not be allocated or freed from exception handling code.

### INPUTS

signalNum - the signal number to free {0..31}

## FreeSprite

### NAME

FreeSprite -- return sprite for use by others and virtual  
sprite machine

### SYNOPSIS

```
FreeSprite(pick)
 d0
```

### FUNCTION

Marks sprite as available for others to use.

### INPUTS

pick = 0-7

### RESULTS

Sprite made available for subsequent callers of GetSprite  
as well as use by Virtual Sprite Machine

### BUGS

These sprite routines are provided to ease sharing of sprite  
hardware and to handle simple cases of sprite usage and  
movement. It is assumed the programs that use these routines  
do want to be good citizens in their hearts (i.e., that they will  
not FreeSprite unless they actually own the sprite).  
Virtual Sprite machine may ignore simple sprite machine.

### SEE ALSO

sprite.h, GetSprite, ChangeSprite, MoveSprite

## FreeSysRequest

### NAME

FreeSysRequest -- free up memory used by a call to  
BuildSysRequest()

### SYNOPSIS

```
FreeSysRequest(Window)
 A0
```

### FUNCTION

This routine frees up all memory allocated by a successful  
call to the BuildSysRequest() procedure. If BuildSysRequest()  
returned a pointer to a Window structure, then your  
program can Wait() for the message port of that window to  
detect an event that satisfies the requester. When you want  
to remove the requester, you call this procedure. It ends  
the requester and deallocates any memory used in the creation  
of the requester.

NOTE: If BuildSysRequest() did not return a pointer to a  
window, you should not call FreeSysRequest().

### INPUTS

Window = a copy of the window pointer returned by a successful  
call to the BuildSysRequest() procedure.

### RESULT

None

### BUGS

None

### SEE ALSO

BuildSysRequest(), Wait(), AutoRequest()

## FreeTrap

### NAME

FreeTrap -- free a processor trap

### SYNOPSIS

```
FreeTrap(trapNum)
 DO
```

### FUNCTION

This function frees a previously allocated trap number for reuse. This call must be performed while running in the same task in which the trap was allocated.

### WARNING

Traps may not be allocated or freed from exception handling code.

### INPUTS

trapNum - the trap number to free [of 0..15]

## FreeVPortCopLists

### NAME

FreeVPortCopLists -- deallocate all intermediate Copper lists and their headers from a viewport

### SYNOPSIS

```
FreeVPortCopLists(viewport)
```

### FUNCTION

Recursively searches display, color, sprite, and user Copper lists and calls FreeMem() to deallocate them from memory

### INPUTS

viewport = pointer to ViewPort structure

### RESULTS

```
vp->DspIns == NULL; vp->SprIns == NULL; vp->ClrIns == NULL;
vp->UCopIns == NULL;
```

### BUGS

none known

### SEE ALSO



## FreeWbObject

### NAME

FreeWbObject -- free all memory in a Workbench object

### SYNOPSIS

```
FreeWbObject(obj)
 A0
```

### FUNCTION

This routine frees all memory in a Workbench object, and the object itself. It is implemented via FreeFreeList().

AllocWbObject() takes care of all the initialization required to set up the objects free list.

This routine is intended only for internal users that can track changes to the Workbench.

### INPUTS

free -- a pointer to a FreeList structure

### RESULTS

### EXCEPTIONS

### SEE ALSO

AllocEntry, FreeEntry, AllocWbObject, FreeFreeList

### BUGS

## fsubi

### NAME

fsubi -- subtract two floating-point numbers

### C USAGE

```
fnum3 = fnum1 - fnum2;
 D1 D0
```

### FUNCTION

Accepts two floating-point numbers and returns the arithmetic subtraction of said numbers. Note that this function is called by compiler-generated code, not by a user-generated function call.

### INPUTS

fnum1 - floating-point number  
fnum2 - floating-point number

### RESULT

fnum3 - floating-point number

### BUGS

None

### SEE ALSO

SPSub,

ftsti

NAME

ftsti -- compares a fast floating-point number against the value zero (0.0) and sets the appropriate condition codes

C USAGE

```
if (!fnum) {...}
 Dl
```

FUNCTION

Accepts a floating-point number and returns the condition codes set to indicate the result of a comparison against the value of zero (0.0). Note that this function is called by compiler generated code, not by a user generated function call.

INPUTS

fnum - floating-point number

RESULT

Condition codes set to reflect the following branches:

```
EQ - fnum = 0.0
NE - fnum != 0.0
PL - fnum >= 0.0
MI - fnum < 0.0
```

BUGS

None

SEE ALSO

SPTst,

GetCC

NAME

GetCC -- get condition codes in a 68010 compatible way.

SYNOPSIS

```
conditions = GetCC()
D0
```

FUNCTION

This function provides a means of obtaining the CPU condition codes in a manner that will make 68010 upgrades transparent.

INPUTS

RESULTS

conditions - the 68000/68010 condition codes

## GetCLBuf

### NAME

GetCLBuf -- convert a character list to contiguous data

### SYNOPSIS

```
length = GetCLBuf(cList, buffer, maxLength)
D0 A0 A1 D1
```

### FUNCTION

Move the cList data into the block of memory pointed to by buffer. Exhaust the character list. If a non-destructive peek at the character list is desired, use SubCL. If the cList will no longer be used, remember to FreeCList.

### INPUTS

cList -  
The cList descriptor used to manage this character list, as returned by AllocCList.  
buffer -  
A pointer for the byte data from the character list.  
maxLength -  
The maximum size of buffer.

### RESULTS

length -  
the number of bytes copied into buffer. This is never greater than maxLength.

### EXCEPTIONS

if cList was bigger than maxLength, the cList is not empty.

## GetCLChar

### NAME

GetCLChar -- get a byte from the beginning of a character list

### SYNOPSIS

```
byte = GetCLChar(cList)
D0 A0
```

### FUNCTION

Get a byte from the beginning of the character list described by the cList.

### INPUTS

cList -  
The cList header used to manage this character list, as returned by AllocCList or StrToCL.

### RESULTS

byte -  
The byte from the beginning of the character list. If no data is available, the upper three bytes are set (longword is -1).

## GetCLWord

### NAME

GetCLWord -- get a word from the beginning of a character list

### SYNOPSIS

```
word = GetCLWord(cList)
D0 A0
```

### FUNCTION

Get a word from the beginning of the character list described by the cList.

### INPUTS

cList -  
The cList header used to manage this character list, as returned by AllocCList or StrToCL.

### RESULTS

word -  
The word from the beginning of the character list. If no data is available, the upper two bytes are set (longword is -1). Partial words (1 byte) are not returned.

## GetColorMap

### NAME

GetColorMap -- allocate and initialize Colormap

### SYNOPSIS

```
cm = GetColorMap(entries)
d0 d0
```

### INPUTS

entries = number of entries for this colormap

### RESULT

cm = pointer to an initialized ColorMap structure.

Allocates and initializes the required structures to be attached to the ViewPort to save color values.  
Returns 0 if cannot allocate memory for structures

### BUGS

### SEE ALSO

SetRGB4 FreeColorMap

## GetDefPrefs

### NAME

GetDefPrefs -- get a copy of the Intuition default Preferences

### SYNOPSIS

```
GetDefPrefs(PrefBuffer, Size)
 A0 D0
```

### FUNCTION

This function gets a copy of the Intuition default Preferences data. It writes the data into the buffer you specify. The number of bytes you want copied is specified by the Size argument.

The default Preferences are those that Intuition uses when it is first opened. If no Preferences file is found, these are the preferences that are used. These would also be the start-up Preferences in an environment that does not use AmigaDOS.

It is legal to take a partial copy of the Preferences structure. The more pertinent Preferences variables have been grouped near the top of the structure to facilitate the memory conservation that can be had by taking a copy of only some of the Preferences structure.

### INPUTS

PrefBuffer = pointer to the memory buffer to receive your copy of the Intuition Preferences.  
Size = the number of bytes in your PrefBuffer—the number of bytes you want copied from the system's internal Preference settings.

### RESULT

Returns your Preferences pointer.

### BUGS

None.

### SEE ALSO

GetPrefs()

## GetDiskObject

### NAME

GetDiskObject -- read in a Workbench disk object

### SYNOPSIS

```
diskobj = GetDiskObject(name)
 D0 A0
```

### FUNCTION

This routine reads in a Workbench disk object in from disk. The name parameter will have a ".info" postpended to it, and the info file of that name will be read. If the call fails, it will return zero. The reason for the failure may be obtained via IoErr().

This routine is very similar to GetIcon, but it shields the programmer from the worst of the grunginess associated with GetIcon. A FreeList structure is allocated just after the DiskObject structure; FreeDiskObject makes use of this to get rid of the memory that was allocated.

### INPUTS

name -- name of the object

### RESULTS

diskobj -- the Workbench disk object in question

### EXCEPTIONS

### SEE ALSO

GetIcon, FreeDiskObject

### BUGS

## GetGBuffers

### NAME

GetGBuffers -- attempts to allocate ALL the buffers of an entire AnimOb

### SYNOPSIS

GetGBuffers(anOb, RPort, db) as called by C  
a0 a1 d0

### FUNCTION

For each sequence of each component of the AnimOb, allocates memory for:

- SaveBuffer
- BorderLine
- CollMask and ImageShadow (point to same buffer)
- if db is set (user wants double-buffering) allocate:
  - DBufPacket
  - BufBuffer

### INPUTS

- a1 = pointer to the AnimOb structure
- a2 = pointer to the current RastPort
- d0 = double-buffer indicator (set TRUE for double-buffering)

### RESULT

TRUE if the memory allocations were all successful, else FALSE

### BUGS

None known

### SEE ALSO

Nothing

## GetIcon

### NAME

GetIcon -- read in a DiskObject structure from disk

### SYNOPSIS

status = GetIcon( name, icon, free )  
D0 A0 A1 A2

### FUNCTION

This routine reads in a DiskObject structure and its associated information. All memory will be automatically allocated, and stored in the specified FreeList. The file name of the info file will be the name parameter with a ".info" postpended to it. If the call fails, a zero will be returned. The reason for the failure may be obtained via IoErr().

Users are encouraged to use GetDiskObject instead of this routine.

### INPUTS

- name -- name of the object
- icon -- a pointer to a DiskObject
- free -- a pointer to a FreeList

### RESULTS

status -- non-zero if the call succeeded.

### EXCEPTIONS

### SEE ALSO

### BUGS

## GetMsg

### NAME

GetMsg -- get next message from a message port

### SYNOPSIS

```
message = GetMsg(port)
DO A0
```

### FUNCTION

This function receives a message from a given message port. It provides a fast, non-copying message receiving mechanism.

The received message is removed from the message port.

This function will not wait. If a message is not present this function will return zero. If a program must wait for a message, it can Wait on the signal specified for the port or use the WaitPort function. There can only be one task waiting for any given port.

Getting the message does not imply that the message is now free to be reused. When the receiver is finished with the message, it may ReplyMsg it.

### INPUT

port - a pointer to the receiver message port

### RESULT

message - a pointer to the first message available. If there are no messages, return zero.

### SEE ALSO

PutMsg, ReplyMsg, WaitPort

## GetPrefs

### NAME

GetPrefs -- get the current setting of the Intuition Preferences

### SYNOPSIS

```
GetPrefs(PrefBuffer, Size)
 A0 D0
```

### FUNCTION

This function gets a copy of the current Intuition Preferences data and writes the data into the buffer you specify. The number of bytes you want copied is specified by the Size argument.

It is legal to take a partial copy of the Preferences structure. The more pertinent Preferences variables have been grouped near the top of the structure to facilitate the memory conservation that can be had by taking a copy of only some of the Preferences structure.

### INPUTS

PrefBuffer = pointer to the memory buffer to receive your copy of the Intuition Preferences.

Size = the number of bytes in your PrefBuffer--the number of bytes you want copied from the system's internal Preference settings.

### RESULT

Returns a copy of your Preferences pointer.

### BUGS

None

### SEE ALSO

GetDefPrefs()

## GetRGB4

### NAME

GetRGB4 -- inquire value of entry in ColorMap

### SYNOPSIS

```
value = GetRGB4(colormap, entry)
D0 A0 D0
```

### INPUTS

colormap = pointer to ColorMap structure.  
entry = index into colormap

### RESULT

Returns -1 if no valid entry  
Return UWORD RGB value 4 bits per gun right justified

### BUGS

### SEE ALSO

SetRGB4 LoadRGB4 GetColorMap FreeColorMap

## GetSprite

### NAME

GetSprite -- attempt to get a sprite for the simple sprite manager.

### SYNOPSIS

```
Sprite_Number = GetSprite(sprite, pick)
d0 a0 d0
```

### FUNCTION

Attempts to allocate one of the eight sprites for private use with the simple sprite manager. This must be done before using further calls to simple sprite machine.

### INPUTS

sprite = ptr to programmers SimpleSprite structure.  
pick = 0-7  
-1 if programmer just wants the next one.

### RESULTS

If pick is 0-7, attempts to allocate the sprite. If the sprite is already allocated, return -1. If pick is -1, allocate the next sprite. If no sprites are available, return -1.

If the sprite is available for allocation, marks it allocated and fill in the 'num' entry of the SimpleSprite structure. If successful, returns the sprite number.

### BUGS

### SEE ALSO

sprite.h FreeSprite ChangeSprite MoveSprite GetSprite



## GetWBObject

### NAME

GetWBObject -- read in a Workbench object

### SYNOPSIS

```
object = GetWBObject(name)
D0 A0
```

### FUNCTION

This routine reads in a Workbench object from disk. The name parameter will have a ".info" postpended to it, and the info file of that name will be read. If the call fails, it will return zero. The reason for the failure may be obtained via IoErr().

This routine is intended only for internal users that can track changes to the Workbench.

### INPUTS

name -- name of the object

### RESULTS

object -- the Workbench object in question

### EXCEPTIONS

### SEE ALSO

### BUGS

## IEEEDPabs

### NAME

IEEEDPabs -- obtain the absolute value of the IEEE double precision floating-point number

### C USAGE

```
fnum1 = IEEEDPabs(fnum2);
D0/D1 D0/D1
```

### FUNCTION

Accepts an IEEE D.P. floating-point number and returns the absolute value of said number.

### INPUTS

fnum2 - IEEE double-precision floating-point number

### RESULT

fnum1 - IEEE double-precision floating-point number

### BUGS

None

### SEE ALSO

## IEEEDPadd

### NAME

IEEEDPadd -- add two IEEE double-precision floating-point numbers

### C USAGE

```
fnum1 = IEEEDPadd(fnum2, fnum3);
D0/D1 D0/D1 D2/D3
```

### FUNCTION

Accepts two IEEE D.P. floating-point numbers and returns the arithmetic sum of said numbers.

### INPUTS

fnum2 - IEEE double-precision floating-point number  
fnum3 - IEEE double-precision floating-point number

### RESULT

fnum1 - IEEE double-precision floating-point number

### BUGS

None

SEE ALSO

## IEEEDPCmp

### NAME

IEEEDPCmp -- compare two IEEE D.P. floating-point numbers and return a relative value indicator

### C USAGE

```
if (IEEEDPCmp(fnum1, fnum2)) {...}
D0/D1 D2/D3
```

### FUNCTION

Accepts two IEEE double-precision floating-point numbers and returns the CCR and the integer functional result as an indicator of the result of said comparison.

### INPUTS

fnum1 - IEEE double-precision floating-point number  
fnum2 - IEEE double-precision floating-point number

### RESULT

Condition codes set to reflect the following branches:

|      |                 |                          |
|------|-----------------|--------------------------|
| LT   | - fnum1 < fnum2 | (Functional Result = -1) |
| GT   | - fnum1 > fnum2 | (Functional Result = +1) |
| ELSE | - fnum1 = fnum2 | (Functional Result = 0)  |

### BUGS

None

SEE ALSO

IEEEEDPDiv

NAME

IEEEEDPDiv -- divide two IEEE double-precision floating-point numbers

C USAGE

```
fnum1 = IEEEEDPMul(fnum2, fnum3);
D0/D1 D0/D1 D2/D3
```

FUNCTION

Accepts two IEEE double-precision floating-point numbers and returns the arithmetic division of said numbers.

INPUTS

fnum2 - IEEE double-precision floating-point number  
fnum3 - IEEE double-precision floating-point number

RESULT

fnum1 - IEEE double-precision floating-point number

BUGS

None

SEE ALSO

IEEEEDPFlt

NAME

IEEEEDPFlt -- convert integer number to IEEE D.P. floating-point

C USAGE

```
fnum = IEEEEDPFlt(inum);
D0/D1 D0
```

FUNCTION

Accepts an integer and returns the converted IEEE double precision floating-point result of said number.

INPUTS

inum - signed integer number

RESULT

fnum - IEEE double-precision floating-point number

BUGS

None

SEE ALSO

IEEEEDPMul

NAME

IEEEEDPMul -- multiply two IEEE double-precision floating-point numbers

C USAGE

```
fnum1 = IEEEEDPMul(fnum2, fnum3);
D0/D1 D0/D1 D2/D3
```

FUNCTION

Accepts two IEEE D.P. floating-point numbers and returns the arithmetic multiplication of said numbers.

INPUTS

fnum2 - IEEE double-precision floating-point number  
fnum3 - IEEE double-precision floating-point number

RESULT

fnum1 - IEEE double-precision floating-point number

BUGS

None

SEE ALSO

IEEEEDPNeg

NAME

IEEEEDPNeg -- negate the supplied IEEE double-precision floating-point number

C USAGE

```
fnum1 = IEEEEDPNeg(fnum2);
D0/D1 D0/D1
```

FUNCTION

Accepts an IEEE D.P. floating-point number and returns the value of said number after having been subtracted from 0.0

INPUTS

fnum2 - IEEE double-precision floating-point number

RESULT

fnum1 - IEEE double-precision floating-point number

BUGS

None

SEE ALSO

IEEEEDPSub

NAME

IEEEEDPSub -- subtract two IEEE double-precision floating-point numbers

C USAGE

```
fnum1 = IEEEEDPSub(fnum2, fnum3);
D0/D1 D0/D1 D2/D3
```

FUNCTION

Accepts two IEEE D.P. floating-point numbers and returns the arithmetic subtraction of said numbers.

INPUTS

fnum2 - IEEE double-precision floating-point number  
fnum3 - IEEE double-precision floating-point number

RESULT

fnum1 - IEEE double-precision floating-point number

BUGS

None

SEE ALSO

IEEEEDPTst

NAME

IEEEEDPTst -- compare an IEEE D.P. floating-point number against the value 0.0 and return a relative value indicator

C USAGE

```
if (IEEEEDPTst(fnum)) {...}
D0/D1
```

FUNCTION

Accepts an IEEE double-precision floating-point number and returns the CCR and the integer functional result as an indicator of the result of comparison against the value 0.0.

NOTE: Using number directly within parenthesis to generate in-line code is much more efficient.

INPUTS

fnum - IEEE double-precision floating-point number

RESULT

Condition codes set to reflect the following branches:

LT - fnum < 0.0 (Functional Result = -1)  
GT - fnum > 0.0 (Functional Result = +1)  
ELSE - fnum = 0.0 (Functional Result = 0)

BUGS

None

SEE ALSO

## IncrCLMark

### NAME

IncrCLMark -- increment a clist mark to the next position

### SYNOPSIS

```
error = IncrCLMark(cList)
D0 A0
```

### FUNCTION

Increment a mark for clist operations to mark the next byte in the clist.

### INPUTS

cList -  
a longword descriptor for a clist that can be used for clist functions.

### RESULTS

error -  
non-zero if the next offset is not in the clist

### EXCEPTIONS

if error is non-zero, the request asked to move the mark beyond the end of the clist, and the mark is invalid.

## Info

### NAME

Info -- Returns information about the disk.

### SYNOPSIS

```
success = Info(.lock, InfoData)
D0 D1 D2
```

### FUNCTION

Info finds out information about any disk in use: 'lock' refers to the disk, or any file on the disk. Info returns the InfoData structure with information about the size of the disk, number of free blocks and any soft errors. Note that InfoData must be longword aligned.

### INPUTS

lock - BCPL pointer to a lock  
InfoData - address of an InfoData structure

### RESULTS

success - boolean

### SPECIAL NOTE:

Note that InfoData must be longword aligned.

InitArea

NAME

InitArea -- Initialize vector collection matrix

SYNOPSIS

```
InitArea(AreaInfo *, buffer *, max vectors)
 a0 al d0
```

FUNCTION

This function provides initialization for the vector collection matrix such that it has a size of (max vectors). The size of the region pointed to by buffer (short pointer) should be five times as large as (max vectors). This size is in bytes. Areafills done by using AreaMove, AreaDraw, and AreaEnd must have enough space allocated in this table to store all the points of the largest fill. If not enough space, the routines will return -1

INPUTS

AreaInfo = pointer to AreaInfo structure  
buffer = pointer to chunk of memory to collect vertices  
max vectors = max number of vectors this buffer can hold

RESULT

Pointers are set up to begin storage of vectors done by AreaMove and AreaDraw.

NOTE

The underlying graphics routines actually split the table into two parts to save coordinates and flags

BUGS

None known.

SEE ALSO

graph.h AreaEnd AreaMove AreaDraw

InitBitMap

NAME

InitBitMap -- initialize bit map structure with input values

SYNOPSIS

```
InitBitMap(bm, depth, width, height)
 a0 d0 d1 d2
```

FUNCTION

Initializes various elements in the BitMap structure to correctly reflect input depth, width, and height. Must be used before use of BitMap in other graphics calls. The Planes[8] are not initialized and need to be set up by the caller. The Planes table was put at the end of the structure so that it may be truncated if needed, as well as extended.

INPUTS

bm = pointer to a BitMap structure (gfx.h)  
depth = number of bitplanes that this bitmap will have  
width = number of bits (columns) wide for this BitMap  
height = number of bits (rows) tall for this BitMap

BUGS

None known.

SEE ALSO

gfx.h

## InitCLPool

NAME  
InitCLPool -- initialize a clist pool

SYNOPSIS  
error = InitCLPool(cLPool, size)  
D0                   A0       D0

FUNCTION  
Initialize a block of memory for use as a pool for clist nodes. This involves setting up a header structure and building a free list of all the nodes.

INPUTS  
cLPool -  
The data area that is to be used as the character list pool for the clist operations.  
size -  
The size of the pool, in bytes. Clist pools are limited to 16M bytes.

RESULTS  
error -  
If the clist pool provided is so small that not even pool management memory will fit, this is set to non-zero.

## InitGels

NAME  
InitGels -- initialize a GEL list; must be called before using GELs

SYNOPSIS  
InitGels(head, tail, GInfo)  
          a0    a1    a2

FUNCTION  
Assigns the VSprites as the head and tail of the GEL list in GfxBase  
Links these two GELs together as the keystones of the list  
If the collHandler vector points to some memory array, sets the BORDERHIT vector to NULL

INPUTS  
head = pointer to the VSprite structure to be used as the GEL list head  
tail = pointer to the VSprite structure to be used as the GEL list tail  
GInfo = pointer to the GelsInfo structure to be initialized

RESULT  
Nothing

BUGS  
None known

SEE ALSO  
Nothing



## InitMasks

### NAME

InitMasks -- initialize all the masks of an AnimOb

### SYNOPSIS

InitMasks(anOb) as called by C  
a0

### FUNCTION

For every sequence of every component, calls InitMasks

### INPUTS

a1 = pointer to the AnimOb

### RESULT

.Nothing

### BUGS

None known

### SEE ALSO

Nothing

## InitLayers

### NAME

InitLayers -- Initialize Layer\_Info structure

### SYNOPSIS

InitLayers(li)  
a0

### INPUTS

li = pointer to LayerInfo structure

### FUNCTION

Initializes Layer\_Info structure in preparation for using other layer operations on this list of layers. Makes the layers unlocked (open).

### SEE ALSO

layers.h

## InitMasks

### NAME

InitMasks -- initialize the BorderLine and CollMask masks of a VSprite

### SYNOPSIS

InitMasks(VS)            as called by C  
                  a0

### FUNCTION

Creates the appropriate BorderLine and CollMask masks of the VSprite  
Correctly detects if the VSprite is actually a Bob definition, handles  
the image data accordingly.

### INPUTS

VS = pointer to the VSprite structure

### RESULT

Nothing

### BUGS

### SEE ALSO

Nothing

## InitRastPort

### NAME

InitRastPort -- Initialize raster port structure

### SYNOPSIS

InitRastPort( rp )  
                  a1

### FUNCTION

Initializes a RastPort structure to standard values.

The struct Rastport describes a control structure for a write-able raster. The RastPort structure describes how a complete single playfield display will be written into. A RastPort structure is referenced whenever any drawing or filling operations are to be performed on a section of memory.

The section of memory that is being used in this way may or may not be presently a part of the current actual on-screen display memory. The name of the actual memory section that is linked to the RastPort is referred to here as a "raster" or as a bitmap.

NOTE: Calling the routine InitRastPort only establishes various defaults. It does NOT establish where, in memory, the rasters are located. To do graphics with this RastPort, the user must set up the BitMap pointer in the RastPort.

### INPUTS

rp = pointer to a RastPort structure.

### RESULT

All entries in RastPort get zeroed out. Exceptions:  
The following get -1:  
Mask, FgPen, AOLPen, LinePtrn  
DrawMode = JAM2  
The font is set to the standard system font.

### BUGS

None known.

### SEE ALSO

rastport.h

## InitRequester

### NAME

InitRequester -- initialize a Requester structure

### SYNOPSIS

```
InitRequester(Requester)
 A0
```

### FUNCTION

The original text for this function was:

This function initializes a requester for general use. After calling InitRequester(), you need fill in only those requester values that fit your needs. The other values are set to states that Intuition regards as NULL.

All this routine actually does is fill the specified Requester structure with zeros. There is no requirement to call this routine before using a Requester structure. For the sake of backward compatibility, this function call remains, but its sole effect is, and is guaranteed to always be, a zero, a mystery, an enigma.

### INPUTS

Requester = a pointer to a Requester structure

### RESULT

None

### BUGS

None

### SEE ALSO

None

## InitStruct

### NAME

InitStruct -- initialize memory from a table

### SYNOPSIS

```
InitStruct(initTable, memory, size);
 A1 A2 D0-0:16
```

### FUNCTION

Clear a memory area except those words whose data and offset values are provided in the initialization table. This initialization table has byte commands to

|      |   |       |      |      |      |      |       |      |      |    |    |         |      |              |
|------|---|-------|------|------|------|------|-------|------|------|----|----|---------|------|--------------|
| load | a | count | byte | word | long | into | given | next | byte | rp | tr | offset, | once | repetitively |
|------|---|-------|------|------|------|------|-------|------|------|----|----|---------|------|--------------|

Not all combinations are supported. The offset, when specified, is relative to the memory pointer provided (Memory), and is initially zero. The initialization data (InitTable) contains byte commands whose 8 bits are interpreted as follows:

#### ddssnnnn

dd the destination type (and size):  
00 next destination, nnnn is count  
01 next destination, nnnn is repeat  
10 destination offset is next byte, nnnn is count  
11 destination offset is next rp, nnnn is count  
ss the size and location of the source:  
00 long, from the next two aligned words  
01 word, from the next aligned word  
10 byte, from the next byte  
11 ERROR - will cause an ALERT (see below)  
nnnn the count or repeat:  
count the (number+1) of source items to copy  
repeat the source is copied (number+1) times.

initTable commands are always read from the next even byte. Given destination offsets are always relative to memory (A2).

The command 00000000 ends the InitTable stream: use 00010001 if you really want to copy one longword.

24 bit APTR not supported for 68020 compatibility -- use long.

### INPUTS

initTable - the beginning of the commands and data to init Memory with. Must be on an even boundary unless only byte initialization is done.  
memory - the beginning of the memory to initialize. Must be on an even boundary if size is specified.  
size - the size of memory, which is used to clear it before initializing it via the initTable. If size is zero, memory is not cleared before initializing. Size is rounded down to the nearest even number before use.

#### IMPLEMENTATION

D0 clear size, command, count and repeat  
D1 destination offset, command type  
A0 current Memory pointer  
A1 current InitTable pointer

D0,D1,A0,A1 destroyed

#### InitTmpRas

##### NAME

InitTmpRas -- Initialize area of local memory for usage by  
areafill, floodfill, text

##### SYNOPSIS

```
InitTmpRas(tmpRas *,buffer *, size)
 a0 al d0
```

##### FUNCTION

The area of memory pointed to by buffer is set up to be used by RastPort routines that may need to get some memory for intermediate operations in preparation to putting the graphics into the final BitMap. tmpRas is used to control the usage of buffer.

##### INPUTS

tmpRas = pointer to a TmpRas structure to be linked into  
a RastPort  
buffer = pointer to a contiguous piece of chip memory.  
size = size in bytes of buffer

##### RESULT

Makes buffer available for users of RastPort

##### BUGS

None known. It would be nice if RastPorts could share one TmpRas.

InitView

NAME

InitView -- initialize View structure

SYNOPSIS

```
InitView(view)
 al
```

FUNCTION

Initializes View structure to default values.

INPUTS

view = pointer to a View structure

RESULT

First, View structure set to all 0s.  
Then values are put in DxOffset,DyOffset to properly position  
default display about .5 inches from top and left on monitor.  
InitView pays no attention to previous contents of view.

BUGS

None known.

SEE ALSO

view.h

InitVPort

NAME

InitVPort -- Initialize ViewPort structure

SYNOPSIS

```
InitVPort(vp)
 a0
```

FUNCTION

Initializes ViewPort structure to default values.

INPUTS

vp = pointer to a ViewPort structure

RESULT

ViewPort structure set to all 0's.

BUGS

None known.

SEE ALSO

view.h

## Input

### NAME

Input -- Identifies the program's initial input file handle.

### SYNOPSIS

```
file = Input()
D0
```

### RESULTS

file - BCPL pointer to a file handle

### FUNCTION

To identify the program's initial input file handle, you use Input.  
(To identify the initial output, see Output.)

## Insert

### NAME

Insert -- insert a node into a list

### SYNOPSIS

```
Insert(list, node, listNode)
A0 A1 A2
```

### FUNCTION

Insert a node into a doubly linked list AFTER a given node position. Insertion at the head of a list is performed by passing a zero value for listNode.

### INPUTS

list - a pointer to the target list.header  
node - the node to insert  
listNode - the node after which to insert

## IntuiTextLength

### NAME

IntuiTextLength -- return the length (pixel width) of an IntuiText

### SYNOPSIS

IntuiTextLength(IText)  
A0

### FUNCTION

This routine accepts a pointer to an instance of an IntuiText structure and returns the length (the pixel width) of the string that is represented by that instance of the structure.

All of the usual IntuiText rules apply. Most notably, if the Font pointer of the structure is set to NULL, you will get the pixel width of your text in terms of the current default font.

### INPUTS

IText = pointer to an instance of an IntuiText structure

### RESULT

Returns the pixel width of the text specified by the IntuiText data.

### BUGS

None

### SEE ALSO

None

## IoErr

### NAME

IoErr -- return extra information from the system

### SYNOPSIS

error = IoErr()  
D0

### RESULTS

error - integer

### FUNCTION

I/O routines return zero to indicate an error. When an error occurs, call this routine to find out more information. Some routines use IoErr(), for example, DeviceProc, to pass back a secondary result.

## IsInteractive

### NAME

IsInteractive -- discover whether a file is connected to a virtual terminal

### SYNOPSIS

```
bool = IsInteractive (file)
 D0 D1
```

### FUNCTION

The function IsInteractive gives a Boolean return. This indicates whether or not the file associated with the file handle 'file' is connected to a virtual terminal.

### INPUTS

file - BCPL pointer to a file handle

### RESULTS

bool - boolean

## ItemAddress

### NAME

ItemAddress -- return the address of the specified MenuItem

### SYNOPSIS

```
ItemAddress(MenuStrip, MenuNumber)
 A0 D0
```

### FUNCTION

This routine feels through the specified MenuStrip and returns the address of the item specified by the MenuNumber. Typically, you will use this routine to get the address of a MenuItem from a MenuNumber sent to you by Intuition after the user has played with your menus.

This routine requires that the arguments be well defined. MenuNumber may be equal to MENUNULL, in which case this routine returns NULL. If MenuNumber does not equal MENUNULL, it is presumed to be a valid item number selector for your MenuStrip, which includes a valid menu number and a valid item number. If the item specified by the above two components has a subitem, the MenuNumber may have a subitem component too.

Note that there must be both a menu number and an item number. Because a subitem specifier is optional, the address returned by this routine may point to either an item or a subitem.

### INPUTS

MenuStrip = a pointer to the first menu in your menu strip.  
MenuNumber = the value that contains the packed data that selects the menu and item (and subitem).

### RESULT

If MenuNumber == MENUNULL, this routine returns NULL. Otherwise, this routine returns the address of the MenuItem specified by MenuNumber.

### BUGS

None

### SEE ALSO

The "Menus" chapter in Amiga Intuition Reference Manual



## LoadRGB4

### NAME

LoadRGB4 -- load RGB color values from table

### SYNOPSIS

```
LoadRGB4(vp, colormap, count)
 a0 a1 d0
```

### FUNCTION

Loads the count words of the colormap from table

### INPUTS

vp = pointer to ViewPort whose colors you want to change  
colormap = pointer to table of RGB values set up like an array  
of USHORTS  
background-- 0x0RGB  
color1 -- 0x0RGB  
color2 -- 0x0RGB  
etc. UWORD per value.  
The colors are interpreted as 15 = maximum intensity.  
0 = minimum intensity.  
count = number of UWORDS in the table to load into the  
colormap starting at color 0 (background) and proceeding  
to the next higher color number

### RESULTS

Store the colors in the ViewPorts colormap. This is a  
table gotten from GetColorMap(number of entries).  
This colormap will be initialized from the Default colormap.

### BUGS

None known

### SEE ALSO

view.h

## LoadSeg

### NAME

LoadSeg -- load a load module into memory

### SYNOPSIS

```
segment = LoadSeg(name)
 D0 D1
```

### FUNCTION

The file 'name' is a load module produced by the linker. LoadSeg takes  
this and scatter loads the code segments into memory, chaining the  
segments together on their first words. It recognizes a zero as  
indicating the end of the chain.

If an error occurs, Loadseg unloads any loaded blocks and returns  
a false (zero) result.

If all goes well (that is, LoadSeg has loaded the module correctly),  
Loadseg returns a pointer to the beginning of the list of blocks.  
Once you have finished with the loaded code, you can unload it with  
a call to UnLoadSeg. (For using the loaded code, see CreateProc.)

### INPUTS

name - address of first character of a null-terminated string

### RESULTS

segment - BCPL pointer to a segment

LoadView

NAME

LoadView -- Use a (possibly freshly created) coprocessor instruction list to create the current display.

SYNOPSIS

LoadView( View )  
A1

FUNCTION

See NAME field. Coprocessor instruction list has been created by InitVPort, MakeView, and MrgCop.

INPUTS

View = a pointer to the View structure, which contains the pointer to the constructed coprocessor instructions list

RESULT

The new View is displayed, according to your instructions. The vertical blank routine will pick this pointer up and direct Copper to start displaying this View.

BUGS

...

SEE ALSO

InitVPort, MakeView, MrgCop  
Intuition's RethinkDisplay()

Lock

NAME

Lock -- lock a directory or file

SYNOPSIS

lock = Lock( name, accessMode )  
D0 D1 D2

FUNCTION

Lock returns, if possible, a filing system lock on the file or directory 'name.' If the accessMode is ACCESS\_READ, the lock is a shared read lock; if the accessMode is ACCESS\_WRITE, it is an exclusive write lock. If LOCK fails (that is, if it cannot obtain a filing system lock on the file or directory) it returns a zero.

Note that the overhead for doing a Lock is less than that for doing an Open. If you want to test to see if a file exists, you should use Lock. Of course, once you've found that it exists, you have to use Open to open it.

INPUTS

name - address of first character of a null-terminated string  
accessMode - integer

RESULTS

lock - BCPL pointer to a lock

## LockLayer

### NAME

LockLayer -- lock layer to make changes to ClipRects

### SYNOPSIS

```
LockLayer(li, l)
 a0 al
```

### INPUTS

li = pointer to LayerInfo structure  
l = pointer to a layer

### FUNCTION

Makes this layer unavailable for other tasks to use.  
If another task is already using this layer, waits for  
it to complete and then takes the layer.

### SEE ALSO

layers.h

## LockLayerInfo

### NAME

LockLayerInfo -- lock the LayerInfo structure.

### SYNOPSIS

```
LockLayerInfo(li)
 a0
```

### INPUTS

li = pointer to LayerInfo structure

### FUNCTION

After the operation that required a LockLayerInfo is complete,  
unlocks the LayerInfo structure so that other tasks may  
affect the layers.

### SEE ALSO

layers.h LockLayerInfo()

## LockLayerRom

NAME LockLayerRom -- lock layer structure by rom(gfx lib) code

SYNOPSIS  
LockLayerRom( layer )  
                  a5

FUNCTION  
Returns when the layer is locked and no other caller may alter the ClipRect structure in the Layer structure.

INPUTS  
layer = pointer to Layer structure

NOTE  
This call does not destroy any registers.  
This call nests so that callers in this chain will not lock themselves out.

Caveat: This lock does not prevent another task from calling LockLayerRom() and not blocking.  
This is potentially dangerous in the case of ScrollRaster which will resort the list of ClipRects although it does not add any new ClipRects or remove any ClipRects.

SEE ALSO  
layers.h

## LockLayers

NAME LockLayers -- lock all layers from graphics output

SYNOPSIS  
LockLayers( li )  
                  a0

INPUTS  
li = pointer to LayerInfo structure

FUNCTION  
First, calls LockLayerInfo.  
Makes all layers in this layer list locked.

SEE ALSO  
layers.h LockLayer() LockLayerInfo()

## MakeLibrary

## NAME

MakeLibrary -- construct a library

## SYNOPSIS

```
library = MakeLibrary(vectors, structure, init, dataSize, segList)
D0 A0 A1 A2 D0 D1
```

## FUNCTION

This function is used for constructing a library vector and data area. Space for the library is allocated from the system's free memory pool. The size fields of the library are filled. The data portion of the library is initialized. A library specific entrypoint is called (init) if present.

## INPUTS

vectors - pointer to an array of function pointers or function displacements. If the first word of the array is -1, then the array contains relative word displacements (based off of vectors); otherwise, the array contains absolute function pointers.

structure - points to an "InitStruct" data region. If null, then it will not be called.

init - an entry point that will be called before adding the library to the system. If null, it will not be called. When it is called, it will be called with the libAddr in D0, and its result will be the result of this function.

dSize - the size of the library data area, including the standard library node data.

segList - pointer to a memory segment list (used by DOS)  
This is passed to a library's init code.

## RESULT

library - the reference address of the library. This is the address used in references to the library, not the beginning of the memory area allocated.

## EXCEPTION

If the library vector table require more system memory than is available, this function will cause a system panic.

## SEE ALSO

InitStruct

## MakeScreen

## NAME

MakeScreen -- do an Intuition-integrated MakeVPort() of a custom screen

## SYNOPSIS

```
MakeScreen(Screen)
A0
```

## FUNCTION

This procedure allows you to do a MakeVPort() for the ViewPort of your custom screen in an Intuition-integrated way. This allows you to do your own screen manipulations without worrying about interference with Intuition's usage of the same ViewPort.

After calling this routine, you can call RethinkDisplay() to incorporate the new ViewPort of your custom screen into the Intuition display.

## INPUTS

Screen = address of the Screen structure.

## RESULT

None

## BUGS

None

## SEE ALSO

RethinkDisplay(), RemakeDisplay(), MakeVPort()

## MakeVPort

### NAME

MakeVPort -- generate display Copper list

### SYNOPSIS

```
MakeVPort(view, viewport)
 a0 al
```

### FUNCTION

Using information in the View and ViewPort constructs intermediate Copper list for this ViewPort.

### INPUTS

view = pointer to View structure  
viewport= pointer to ViewPort structure  
The viewport must have valid ptr to RasInfo

### RESULTS

Constructs intermediate Copper list and puts pointers in viewport.DspIns  
If the ColorMap ptr in ViewPort is nil, it uses colors from the default color table.  
If DUALPF in Modes, there must be a second RasInfo pointed to by the first RasInfo

### BUGS

### SEE ALSO

MrgCop() view.h  
Intuition's MakeScreen(), RemakeDisplay(), and RethinkDisplay()

## MarkCList

### NAME

MarkCList -- mark a position in a clist

### SYNOPSIS

```
error = MarkCList(cList, offset)
 A0 D0
```

### FUNCTION

Mark the clist for index operations by specifying a byte offset into the clist. Note that only one mark is retained by each clist. If the byte to which the mark refers is subsequently manipulated, the mark will become invalid.

### INPUTS

cList - a longword descriptor for a clist that can be used for clist functions.  
offset - a byte offset into the clist. The first byte in the clist is at offset zero. This value should not be greater than (SizeCList-1).

### RESULTS

error - non-zero if the offset is not in the clist

### EXCEPTIONS

if the offset is more than the length of the clist, the mark is invalid.

## MatchToolValue

### NAME

MatchToolValue -- check a tool type variable for a particular value

### SYNOPSIS

```
result = MatchToolValue(typeString, value)
D0 A0 A1
```

### FUNCTION

MatchToolValue is useful for parsing a tool type value for a known value. It knows how to parse the syntax for a tool type value (in particular, it knows that '|' separates alternate values).

### INPUTS

typeString - a ToolType value (as returned by FindToolType)  
value - you are interested if value appears in typeString

### RESULTS

result - a one if the value was in typeString

### EXCEPTIONS

### EXAMPLE

Assume there are two type strings:  
type1 = "text"  
type2 = "a|b|c"

```
MatchToolValue(type1, "text") returns 1
MatchToolValue(type1, "data") returns 0
MatchToolValue(type2, "a") returns 1
MatchToolValue(type2, "b") returns 1
MatchToolValue(type2, "d") returns 0
MatchToolValue(type2, "a|b") returns 0
```

### SEE ALSO

FindToolType

### BUGS

## ModifyIDCMP

### NAME

ModifyIDCMP -- modify the state of the window's IDCMP

### SYNOPSIS

```
ModifyIDCMP(Window, IDCMPFlags)
A0 D0
```

### FUNCTION

This routine modifies the state of your window's IDCMP (Intuition Direct Communication Message Port). The state is modified to reflect your desires as described by the flag bits in the value IDCMPFlags. If the IDCMPFlags argument equals NULL, you are asking for the ports to be closed; if they are open, they will be closed. If you set any of the IDCMPFlags, this means that you want the message ports to be open; if not currently open, the ports will be opened.

The four actions that might be taken are described below:

- o If there is currently no IDCMP in the given window and IDCMPFlags is NULL, nothing happens.
- o If there is currently no IDCMP in the given window and any of the IDCMPFlags are selected (set), the IDCMP of the window is created, including allocating and initializing the message ports and allocating a signal bit for your port. See "Input and Output Methods" in the Amiga Intuition Reference Manual for full details.
- o If the IDCMP for the given window is opened and the IDCMPFlags argument is NULL, Intuition will close the ports, free the buffers, and free your signal bit. The current task must be the same one that was active when this signal bit was allocated.
- o If the IDCMP for the given window is opened and the IDCMPFlags argument is not NULL, this means that you want to change which events will be broadcast to your program through the IDCMP.

NOTE: You can set up the Window->UserPort to any port of your own before you call ModifyIDCMP(). If IDCMPFlags is non-null but your UserPort is already initialized, Intuition will assume that it is a valid port with task and signal data preset and will not disturb your set-up; Intuition will just allocate the Intuition message port for your window. The converse is true as well; if UserPort is NULL when you call here with IDCMPFlags == NULL, only the Intuition port will be deallocated. This allows you to use a port that you already have allocated:

- o OpenWindow() with IDCMPFlags equal to NULL (open no ports).
- o Set the UserPort variable of your window to any valid port of your own choosing.
- o Call ModifyIDCMP() with IDCMPFlags set to what you want.
- o Then, to clean up later, set UserPort equal to NULL before calling CloseWindow() (leave IDCMPFlags alone).

A grim, foreboding note: If you are ever rude enough to

close an IDCMP without first having Reply()'d to all of the messages sent to the IDCMP port, Intuition will in turn be so rude as to reclaim and deallocate its messages without waiting for your permission.

INPUTS

Window = pointer to the Window structure containing the IDCMP ports  
IDCMPFlags = the flag bits describing the new desired state of the IDCMP

RESULT

None

BUGS

None

SEE ALSO

OpenWindow()

ModifyProp

NAME

ModifyProp -- modify the current parameters of a proportional gadget

SYNOPSIS

```
ModifyProp(PropGadget, Pointer, Requester,
 A0 A1 A2
 Flags, HorizPot, VertPot, HorizBody, VertBody)
 D0 D1 D2 D3 D4
```

FUNCTION

This routine modifies the parameters of the specified proportional gadget. The gadget's internal state is then recalculated and the imagery is redisplayed.

The Pointer argument can point to either a Window or a Screen structure. Which one it actually points to is decided by examining the SCRGADGET flag of the gadget. If the flag is set, Pointer points to a Screen structure; otherwise, it points to a Window structure.

The Requester variable can point to a Requester structure. If the gadget has the REQADGET flag set, the gadget is in a requester and the Pointer must necessarily point to a window. If this is not the gadget of a requester, the Requester argument may be NULL.

INPUTS

PropGadget = pointer to the structure of a proportional gadget.  
Pointer = pointer to the structure of the "owning" display element of the gadget, which is a window or a screen.  
Requester = pointer to a Requester structure (this may be NULL if this is not a requester gadget).  
Flags = value to be stored in the Flags variable of the PropInfo.  
HorizPot = value to be stored in the HorizPot variable of the PropInfo.  
VertPot = value to be stored in the VertPot variable of the PropInfo.  
HorizBody = value to be stored in the HorizBody variable of the PropInfo.  
VertBody = value to be stored in the VertBody variable of the PropInfo.

RESULT

None

BUGS

None

SEE ALSO

None



Move

\*

NAME

Move -- move graphics pen position

SYNOPSIS

```
Move(rp, x, y)
 al d0 dl
```

FUNCTION

Moves graphics pen position to (x,y) relative to upper left (0,0) of RastPort.

Note: Text uses the same position.

INPUTS

rp = pointer to a RastPort structure  
x,y= point in the RastPort

MoveLayer

NAME

MoveLayer -- move nonbackdrop layer to new position in BitMap

SYNOPSIS

```
MoveLayer(li, l, dx, dy)
 a0 al d0 dl
```

INPUTS

li = pointer to LayerInfo structure  
l = pointer to a nonbackdrop layer  
dx = delta to add to current x position  
dy = delta to add to current y position

FUNCTION

Moves this layer to new position in shared BitMap:  
If any refresh layers become revealed, collects damage and sets REFRESH bit in layer Flags.

SEE ALSO

layers.h

## MoveLayerInFrontOf

### NAME

MoveLayerInFrontOf -- put layer in front of another layer

### SYNOPSIS

```
BOOLEAN MoveLayerInFrontOf(layertomove, target)
 a0 al
```

### INPUTS

layertomove : layer to moved  
target : move layertomove infront of target

### FUNCTION

Moves this layer in front of target, swapping bits in and out of the display with other layers. If this is a refresh layer, collects damage list and sets bit in Flags if redraw required. By clearing the BACKDROP bit in the layers Flags, you may bring a Backdrop layer up to the front of all other layers.

### RETURNS

TRUE if operation successful  
FALSE if operation unsuccessful (probably out of memory)

### SEE ALSO

layers.h

## MoveScreen

### NAME

MoveScreen -- attempt to move the screen by the delta amounts

### SYNOPSIS

```
MoveScreen(Screen, DeltaX, DeltaY)
 A0 D0 D1
```

### FUNCTION

Attempts to move the specified screen. This movement must follow one constraint (only for the current release of the software): horizontal movements are ignored.

If the DeltaX and DeltaY variables you specify would move the screen in a way that violates the above restriction, the screen will be moved as far as possible.

### INPUTS

Screen = pointer to a Screen structure.  
DeltaX = amount to move the screen on the x axis.  
DeltaY = amount to move the screen on the y axis.

### RESULT

None

### BUGS

None

### SEE ALSO

None

## MoveSprite

### NAME

MoveSprite -- move sprite to a point relative to top of viewport

### SYNOPSIS

```
MoveSprite(vp, sprite, x, y)
 a0 al d0 dl
```

### FUNCTION

Moves sprite image to new place on display.

### INPUTS

vp = pointer to ViewPort structure  
= 0, if sprite positioned relative to View  
sprite = pointer to SimpleSprite structure  
x,y = new position relative to top of viewport

### RESULTS

### BUGS

### SEE ALSO

sprite.h FreeSprite ChangeSprite GetSprite

## MoveWindow

### NAME

MoveWindow -- ask Intuition to move a window

### SYNOPSIS

```
MoveWindow(Window, DeltaX, DeltaY)
 A0 D0 D1
```

### FUNCTION

This routine sends a request to Intuition asking to move the window the specified distance. The delta arguments describe how far to move the window along the respective axes. Note that the window will not be moved immediately; it will be moved the next time Intuition receives an input event, which happens currently at a minimum rate of ten times per second and a maximum of sixty times a second.

This routine does no error-checking. If your delta values specify some far corner of the universe, Intuition will attempt to move your window to the far corners of the universe. Because of the distortions in the space-time continuum that can result from this, as predicted by special relativity, the result is generally not a pretty sight.

### INPUTS

Window = pointer to the structure of the window to be moved.  
DeltaX = signed value describing how far to move the window on the x axis.  
DeltaY = signed value describing how far to move the window on the y axis.

### RESULT

None

### BUGS

None

### SEE ALSO

SizeWindow(), WindowToFront(), WindowToBack()

## MrgCop

### NAME

MrgCop -- Merge together coprocessor instructions.

### SYNOPSIS

```
MrgCop(View)
 Al
```

### FUNCTION

Merge together the display, color, sprite and user coprocessor instructions into a single coprocessor instruction stream. This essentially creates a per-display-frame program for the coprocessor. This function MrgCop is used, for example, by the graphics animation routines which effectively add information into an essentially static background display. This changes some of the user or sprite instructions, but not those which have formed the basic display in the first place. When all forms of coprocessor instructions are merged together, you will have a complete per-frame instruction list for the coprocessor.

Restrictions: Each of the coprocessor instruction lists MUST be internally sorted in min to max Y-X order. The merge routines depend on this! Each list must be terminated using CEND(Copper list)

### INPUTS

View - a pointer to the view structure whose coprocessor instructions are to be merged.

### RESULT

The View structure will now contain a complete, sorted/merged list of instructions for the coprocessor, ready to be used by the display processor. The display processor is told to use this new instruction stream through the instruction LoadView().

### BUGS

...

### SEE ALSO

InitVPort, MrgCop, LoadView  
Intuition's RethinkDisplay()

## NewLayerInfo

### NAME

NewLayerInfo -- allocate and Initialize full Layer\_Info structure

### SYNOPSIS

```
NewLayerInfo()
```

### INPUTS

None

### FUNCTION

Allocates memory required for full Layer\_Info structure. Initializes Layer\_Info structure in preparation to use other layer operations on this list of layers. Makes the layers unlocked (open).

### RETURNS

pointer to Layer\_Info structure if successful  
NULL if not enough memory

### SEE ALSO

layers.h

## NewRegion

### NAME

NewRegion -- get a region of size 0

### SYNOPSIS

```
rgn = (struct Region *)NewRegion()
d0
```

### Function

Create a Region structure, initialize it to empty and return a pointer it.

### INPUTS

none

### BUGS

## OffGadget

### NAME

OffGadget -- disable the specified gadget

### SYNOPSIS

```
OffGadget(Gadget, Pointer, Requester)
A0 A1 A2
```

### FUNCTION

This command disables the specified gadget. When a gadget is disabled, these things happen:

- o Its imagery is displayed ghosted.
- o The GADGDISABLED flag is set.
- o The gadget cannot be selected by the user.

The Pointer argument must point to a Window structure. The Requester variable can point to a Requester structure. If the gadget has the REQGADGET flag set, the gadget is in a requester and Pointer must necessarily point to the window containing that requester. If this is not the gadget of a requester, the Requester argument may be NULL.

NOTE: It is never safe to tinker with the gadget list yourself. Do not supply some gadget that Intuition has not already processed in the usual way.

NOTE: If you have specified that this is a gadget of a requester, that requester must be currently displayed.

### INPUTS

Gadget = pointer to the structure of the gadget that you want disabled.

Pointer = pointer to a Window structure.

Requester = pointer to a Requester structure (may be NULL if this is not a requester gadget list).

### RESULT

None

### BUGS

None

### SEE ALSO

OnGadget()

## OffMenu

### NAME

OffMenu -- disable the given menu or menu item

### SYNOPSIS

OffMenu(Window, MenuNumber)  
A0 D0

### FUNCTION

This command disables a subitem, an item, or a whole menu. If the base of the menu number matches the menu currently revealed, the menu strip is redisplayed.

### INPUTS

Window = pointer to the Window structure.  
MenuNumber = the menu piece to be enabled.

### RESULT

None

### BUGS

None

### SEE ALSO

OnMenu()

## OnGadget

### NAME

OnGadget -- enable the specified gadget

### SYNOPSIS

OnGadget(Gadget, Pointer, Requester)  
A0 A1 A2

### FUNCTION

This command enables the specified gadget. When a gadget is enabled, these things happen:

- o Its imagery is displayed normally (not ghosted).
- o The GADGDISABLED flag is cleared.
- o The gadget can thereafter be selected by the user.

The Pointer argument must point to a Window structure. The Requester variable can point to a Requester structure. If the gadget has the REQGADGET flag set, the gadget is in a requester and Pointer must point to the Window containing the requester. If this is not the gadget of a requester, the requester argument may be NULL.

NOTE: It is never safe to tinker with the gadget list yourself. Do not supply some gadget that Intuition has not already processed in the usual way.

NOTE: If you have specified that this is a gadget of a requester, that requester must be currently displayed.

### INPUTS

Gadget = pointer to the structure of the gadget that you want enabled.

Pointer = pointer to a Window structure.

Requester = pointer to a Requester structure (may be NULL if this is not a requester gadget list).

### RESULT

None

### BUGS

None

### SEE ALSO

OffGadget()

## OnMenu

### NAME

OnMenu -- enable the given menu or menu item

### SYNOPSIS

```
OnMenu(Window, MenuNumber)
 A0 D0
```

### FUNCTION

This command enables a subitem, an item, or a whole menu. If the base of the menu number matches the menu currently revealed, the menu strip is redisplayed.

### INPUTS

Window = pointer to the window.  
MenuNumber = the menu piece to be enabled.

### RESULT

None

### BUGS

None

### SEE ALSO

OffMenu()

## Open

### NAME

Open -- open a file for input or output

### SYNOPSIS

```
file = Open(name, accessMode)
 D0 D1 D2
```

### FUNCTION

Open opens 'name' and returns a file handle. If the accessMode is MODE\_OLDFILE (=1005), OPEN opens an existing file for reading or writing.

However, Open creates a new file for writing if the value is MODE\_NEWFILE (=1006). The 'name' can be a filename (optionally prefaced by a device name), a simple device such as NIL:, a window specification such as CON: or RAW: followed by window parameters, or \*, representing the current window.

For further details on the devices NIL:, CON:, and RAW:, see chapter 1 of the of the AmigaDOS User's Manual. If Open cannot open the file 'name' for some reason, it returns the value zero (0). In this case, a call to the routine IoErr() supplies a secondary error code.

For testing to see if a file exists, see the entry under Lock.

### INPUTS

name - address of first character of a null-terminated string  
accessMode - integer

### RESULTS

file - BCPL pointer to file handle

## OpenDevice

### NAME

OpenDevice -- gain access to a device

### SYNOPSIS

```
error = OpenDevice(devName, unitNumber, iORequest, flags)
D0 A0 D0 A1 D1
```

### FUNCTION

This function opens the named device/unit and initializes the given I/O request block.

### INPUTS

devName - requested device name

unitNumber - the unit number to open on that device. The format of the unit number is device specific.

iORequest - the I/O request block to be returned with appropriate fields initialized.

flags - additional driver specific information. This is sometimes used to request opening a device with exclusive access.

### RESULTS

error - zero if successful, else an error is returned

### SEE ALSO

CloseDevice

## OpenDiskFont

### NAME

OpenDiskFont - load and get a pointer to a disk font

### SYNOPSIS

```
font = OpenDiskFont(textAttr)
D0 A0
```

### FUNCTION

This function finds the font with the specified textAttr on disk, loads it into memory, and returns a pointer to the font that can be used in subsequent SetFont() and CloseFont() calls. It is important to match this call with a corresponding CloseFont() call for effective management of font memory.

If the font is already in memory, the copy in memory is used. The disk copy is not reloaded.

### INPUTS

textAttr = a TextAttr structure that describes the text font attributes desired.

### EXCEPTIONS

D0 is zero if the desired font cannot be found.



## OpenFont

### NAME

OpenFont -- get a pointer to a system font.

### SYNOPSIS

```
font = OpenFont(textAttr), graphicsLib
D0 A0 A6
```

### FUNCTION

This function searches the system font space for the graphics text font that best matches the attributes specified. The pointer to the font returned can be used in subsequent SetFont and CloseFont calls. It is important to match this call with a corresponding CloseFont call for effective management of RAM fonts.

### INPUTS

textAttr - a TextAttr structure that describes the text font attributes desired

### EXCEPTIONS

D0 is zero if the desired font cannot be found. If the named font is found, but the size and style specified are not available, a font with the nearest attributes is returned.

## OpenLibrary

### NAME

OpenLibrary -- gain access to a library

### SYNOPSIS

```
library = OpenLibrary(libName, version)
D0 A1 D0
```

### FUNCTION

This function returns a pointer to a library that was previously installed into the system. If the requested library is exists, and if the library version is greater than or equal to the requested version, then the open will succeed.

### INPUTS

libName - the name of the library to open  
version - the version of the library required.

### RESULTS

library - a library pointer for a successful open, else zero

### SEE ALSO

CloseLibrary

## OpenResource

### NAME

OpenResource -- gain access to a resource

### SYNOPSIS

```
resource = OpenResource(resName)
D0 A1
```

### FUNCTION

This function returns a pointer to a resource that was previously installed into the system.

### INPUTS

resName - the name of the resource requested.

### RESULTS

resource - if successful, a resource pointer, else null

### SEE ALSO

CloseResource

## OpenScreen

### NAME

OpenScreen -- open an Intuition screen

### SYNOPSIS

```
OpenScreen(NewScreen)
 A0
```

where the NewScreen structure is initialized with:

Left, Top, Width, Height, Depth, DetailPen, BlockPen, ViewModes, Type, Font, DefaultTitle, Gadgets

### FUNCTION

This command opens an Intuition screen according to the specified parameters. It does all the allocations, sets up the screen structure and all substructures completely, and links this screen's ViewPort into Intuition's View of the world.

Before you call OpenScreen(), you must initialize an instance of a NewScreen structure. NewScreen is a structure that contains all of the arguments needed to open a screen. The NewScreen structure may be discarded immediately after it is used to open the screen.

The TextAttr pointer that you supply as an argument will be used as the default font for all Intuition-managed text that appears in the screen and its windows. This includes, but is not limited to, the text on the title bars of both the screen and windows.

The SHOWTITLE flag is set to TRUE by default when a screen is opened. This causes the screen's title bar to be displayed when the screen first opens. To hide the title bar, you must call the routine ShowTitle().

### INPUTS

NewScreen = pointer to an instance of a NewScreen structure, which is initialized with the following information:

---

LeftEdge = initial x position of your screen (should be zero for now).

TopEdge = initial y position of the opening screen.

Width = the width for this screen's RastPort.

Height = the height for this screen's RastPort.

Depth = number of bit-planes.

DetailPen = pen number for details (such as gadgets or text in the title bar).

BlockPen = pen number for block fills (such as the title bar).

Type = screen type (for any screen not created by Intuition, this should be equal to CUSTOMSCREEN). Types currently supported include only CUSTOMSCREEN, which is your own screen.

You may also set the Type flag CUSTOMBITMAP and then supply your own BitMap for Intuition to use, rather than having Intuition allocate the display memory for

you.

ViewModes = the appropriate flags for the data type ViewPort.Modes. These might include:  
HIRES for this screen to be HIRES width.  
INTERLACE for the display to switch to interlaced mode.  
SPRITES for this screen to use sprites.  
DUALPF for dual-playfield mode.

Font = pointer to the default TextAttr structure for this screen and all windows that open in this screen.

DefaultTitle = pointer to a line of text that will be displayed along the screen's title bar. The text will be null-terminated. If this argument is set to NULL, no text will be produced.

Gadgets = this should be set to NULL.

CustomBitMap = If you're not supplying a custom BitMap, this value is ignored. However, if you have your own display memory that you want used for this screen, the CustomBitMap argument should point to the BitMap that describes your display memory. See the "Screens" chapter in the Amiga Intuition Reference Manual and the "Graphics Primitives" chapter in this manual for more information about BitMaps.

#### RESULT

If all is well, the routine returns the pointer to your new screen. If anything goes wrong, the routine returns NULL.

#### BUGS

None

#### SEE ALSO

OpenWindow(), ShowTitle()

OpenWindow

#### NAME

OpenWindow -- open an Intuition window

#### SYNOPSIS

```
OpenWindow(NewWindow)
 A0
```

where the NewWindow structure is initialized with:

Left, Top, Width, Height, DetailPen, BlockPen, Flags, IDCMPFlags, Gadgets, CheckMark, Text, Type, Screen, BitMap, MinWidth, MinHeight, MaxWidth, MaxHeight

#### FUNCTION

This command opens an Intuition window of the given height, width, and depth, including the specified system gadgets as well as any of your own. It allocates everything you need to get going.

Before you call OpenWindow(), you must initialize an instance of a NewWindow structure, which contains all of the arguments needed to open a window. The NewWindow structure may be discarded immediately after it is used to open the window.

If Type == CUSTOMSCREEN, you must have opened your own screen already via a call to OpenScreen(). Then Intuition uses your Screen argument for the pertinent information needed to get your window going. On the other hand, if Type == one of Intuition's standard screens, your Screen argument is ignored. Instead, Intuition will check to see whether or not that screen already exists; if it does not, it will be opened first before Intuition opens your window in the standard screen. If the flag SUPER\_BITMAP is set, the BitMap variable must point to your own BitMap. The DetailPen and the BlockPen are used for system drawing; for instance, the title bar is first filled using the BlockPen, and then the gadgets and text are drawn using DetailPen. You can supply special pens for your window, or you can use the screen's pens instead (by setting either of these arguments to -1).

#### INPUTS

NewWindow = pointer to an instance of a NewWindow structure, which is initialized with the following data:

LeftEdge = the initial x position for your window.

TopEdge = the initial y position for your window.

Width = the initial width of this window.

Height = the initial height of this window.

DetailPen = pen number (or -1) for the drawing of window details (such as gadgets or text in the title bar).

BlockPen = pen number (or -1) for window block fills (such as the title bar)

Flags = specifiers for your requirements of this window, as follows.

- o System gadgets you want attached to your window:
- o WINDOWDRAG allows this window to be dragged.

- o WINDOWDEPTH lets the user depth-arrange this window.
  - o WINDOWCLOSE attaches the standard close gadget.
  - o WINDOWSIIZING allows this window to be sized. If you ask for the WINDOWSIIZING gadget, you must specify one or both of the flags SIZEBRIGHT and SIZEBOTTOM below; if you do not, the default is SIZEBRIGHT. See the following SIZEBRIGHT and SIZEBOTTOM items for extra information.
  - o SIZEBRIGHT is a special system gadget flag that you set to specify whether or not you want the right border adjusted to account for the physical size of the sizing gadget. The sizing gadget must, after all, take up room in either the right or the bottom border (or both, if you like) of the window. Setting either this or the SIZEBOTTOM flag selects which edge will take up the slack. This will be particularly useful to applications that want to use the extra space for other gadgets (such as a proportional gadget and two Booleans done up to look like scroll bars) or, for instance, applications that want every possible horizontal bit and are willing to lose lines vertically.  
NOTE: If you select WINDOWSIIZING, you must select either SIZEBRIGHT or SIZEBOTTOM or both. If you select neither, the default is SIZEBRIGHT.
  - o SIZEBOTTOM is a special system gadget flag that you set to specify whether or not you want the bottom border adjusted to account for the physical size of the sizing gadget. For details, refer to SIZEBRIGHT above. NOTE: If you select WINDOWSIIZING, you must select either SIZEBRIGHT or SIZEBOTTOM or both. If you select neither, the default is SIZEBRIGHT.
  - o GIMMEZEROZERO produces easy but expensive output.
  - o Type of window raster you want:
    - o SIMPLE\_REFRESH
    - o SMART\_REFRESH
    - o SUPER\_BITMAP
  - o BACKDROP specifies whether or not you want this window to be one of Intuition's special backdrop windows. See BORDERLESS as well.
  - o REPORTMOUSE specifies whether or not you want the program to "listen" to mouse movement events whenever its window is active. If you want to change whether or not your window is listening to the mouse after you have opened your window, you can call ReportMouse(). Whether or not your window is listening to the mouse is also affected by gadgets, because they can cause the program to get mouse movement reports. The reports (either InputEvents or messages on the IDCMP) that you get will have the x,y coordinates of the current mouse position, relative to the upper left corner of your window (GIMMEZEROZERO notwithstanding). This flag can work in conjunction with the IDCMP flag called MOUSEMOVE, which allows your program to listen via the IDCMP.
  - o BORDERLESS should be set if you want a window with no default border padding. Your window may have border padding anyway, depending on the gadgetry you have requested for the window, but you will not get the standard border lines and spacing that come with typical windows. This is a good way to take over the entire screen, since you can have a window cover the entire width of the screen using this flag. This will work particularly well in conjunction with the BACKDROP flag (see above), because it allows you to open a window that fills the entire screen.  
NOTE: This is not a flag that you want to set casually, since it may cause visual confusion on the screen. The window borders are the only dependable visual division between various windows and the background screen. Taking away the border takes away that visual cue, so make sure that your design does not need it before you proceed.
  - o ACTIVATE is the flag you set if you want this window to automatically become the active window. The active window is the one that receives input from the keyboard and mouse. It is usually a good idea to have the window you open when your application first starts up be an ACTIVATED one, but all others opened later should not be ACTIVATED. (If the user is off doing something with another screen, for instance, your new window will change where the input is going, which would have the effect of yanking the input rug from under the user.) Please use this flag thoughtfully and carefully.
  - o RMBTRAP, when set, causes the right mouse button events to be trapped and broadcast as events. Your program can receive these events through either the IDCMP or the console.
- IDCMPFlags = IDCMP is the acronym for Intuition Direct Communications Message Port. If any of the IDCMP flags is selected, Intuition will create a pair of message ports and use them for direct communications with the task that is opening this window (as compared with broadcasting information via the console device). See the "Input and Output Methods" chapter of "Amiga Intuition Reference Manual" for complete details.
- You request an IDCMP by setting any of these flags. Except for the special "verify" flags, every other flag you set tells Intuition that if a given event occurs that your program wants to know about, Intuition should broadcast the details of that event through the IDCMP rather than via the console device. This allows a program to interface with Intuition directly, rather than going through the console device.
- Remember, if you are going to open both an IDCMP and a console, it will be far better to get most of the event messages via the console. Reserve your usage of the IDCMP for special performance cases; that is, when you are not going to open a console for your window and yet you do want to learn about a certain set of events (for instance, CLOSEWINDOW); another example is SIZEVERIFY, which is a function that you get only through the use

of the IDCMP (because the console does not give you any way to talk to Intuition directly). On the other hand, if the IDCMPFlags argument is equal to zero, no IDCMP is created and the only way you can learn about any window event for this window is via a console opened for this window. For instance, you have no way to SIZEVERIFY.

If you want to change the state of the IDCMP after you have opened the window (including opening or closing the IDCMP), you call the routine ModifyIDCMP().

The flags you can set are explained below:

- o REQVERIFY is a flag that, like SIZEVERIFY and MENUVERIFY (see below), specifies that you want to make sure that your graphical state is quiescent before something extraordinary happens, such as the drawing of a rectangle of graphical data in your window. If you are drawing in that window, you probably will wish to make sure that you have ceased drawing before the user is allowed to bring up the DMRequest you have set up. The same goes for when the system has a requester for the user. Set this flag to ask for that verification step.
- o REQCLEAR is the flag you set to get notification when the last requester is cleared from your window and it is safe for you to start output again (presuming that you are using REQVERIFY).
- o REQSET is a flag that you set to receive a broadcast when the first requester is opened in your window. Compare this with REQCLEAR above. This function is distinct from REQVERIFY. REQSET merely tells your program that a requester has opened, whereas REQVERIFY requires the program to respond before the requester is opened.
- o MENUVERIFY is the flag you set to have Intuition stop and wait for your program to finish all graphical output to the window before drawing the menus. Menus are currently drawn in the most memory-efficient way, which involves interrupting output to all windows in the screen before the menus are drawn. If you need to finish your graphical output before this happens, you can set this flag to make sure that you do.
- o SIZEVERIFY is used when the program sends output to the window that depends on a knowledge of the current size of the window. If the user wants to resize the window, you may want to make sure that any queued output completes before the sizing takes place (critical text, for instance). To do so, set this flag. Then, when the user wants to size, Intuition will send the program the SIZEVERIFY message and Wait() until the program replies that it is all right to proceed with the sizing.  
NOTE: Saying that Intuition will Wait() until your program replies is really saying that the user will wait until the program replies, which suffers the great negative potential of user-unfriendliness. Remember to use this flag sparingly, and, as always with any IDCMP message your program receives, reply promptly! After the user has sized the window, your

program can find out about it by using NEWSIZE.

- o NEWSIZE is the flag that tells Intuition to send an IDCMP message after the user has resized your window. At this point, you could examine the size variables in your Window structure to discover the new size of the window.
- o REFRESHWINDOW, when set, will cause a message to be sent whenever your window needs refreshing. This flag makes sense only with SIMPLE\_REFRESH and SMART\_REFRESH windows.
- o MOUSEBUTTONS will make sure your program receives reports about mouse-button up/down events. NOTE: Only the events that mean nothing to Intuition are reported. If the user clicks the select button over a gadget, Intuition deals with it without sending any message.
- o MOUSEMOVE works only if you set the REPORTMOUSE flag (see above) or if one of your gadgets has the flag FOLLOWMOUSE set. Then all mouse movements will be reported through the IDCMP.
- o GADGETDOWN specifies that when the user "selects" a gadget you have created with the GADGIMMEDIATE flag set, the fact will be broadcast through the IDCMP.
- o GADGETUP specifies that when the user "releases" a gadget that you have created with the RELVERIFY flag set, the fact will be broadcast through the IDCMP.
- o MENUPICK specifies that MenuNumber data be sent to your program.
- o CLOSEWINDOW specifies that the CLOSEWINDOW event be broadcasted through the IDCMP rather than the console device.
- o RAWKEY specifies that all RAWKEY events be transmitted via the IDCMP. Note that these are absolutely raw keycodes, which you will have to massage before using. Setting this and the MOUSE flags effectively eliminates the need to open a console device to get input from the keyboard and mouse. Of course, in exchange you lose all of the console features, most notably the "cooking" of input data and the systematic output of text to your window.
- o VANILLAKEY is the raw keycode RAWKEY event translated into the current default character keymap of the console device. In the USA, the default keymap is ASCII characters. When you set this flag, you will get IntuiMessages where the Code field has a character representing the key struck on the keyboard.
- o INTUITICKS gives you simple timer events from Intuition when your window is the active one; it may help you avoid opening and managing the timer device. With this flag set, you will get only one queued-up INTUITICKS message at a time. If Intuition notices that you've been sent an INTUITICKS message and haven't replied to it, another message will not be sent.  
Intuition receives timer events ten times a second (approximately).
- o Set ACTIVEWINDOW and INACTIVIEWINDOW to discover when your window becomes activated or inactivated.

Gadgets = a pointer to the first of a linked list of your own gadgets that you want attached to this window. Can

be NULL if you have no gadgets of your own.

CheckMark = a pointer to an instance of the Image structure that contains the imagery you want used when any of your MenuItem's is to be checkmarked. If you do not want to supply your own imagery and prefer to use Intuition's own checkmark, set this argument to NULL.

Text = a null-terminated line of text that will appear on the title bar of your window (may be NULL if you want no text).

Type = the screen type for this window. If this equals CUSTOMSCREEN, you must have already opened a custom screen (see text above). Types available include:

- o WBENCHSCREEN
- o CUSTOMSCREEN

Screen = if your type is one of Intuition's standard screens, this argument is ignored. However, if type == CUSTOMSCREEN, this must point to the structure of your own screen.

BitMap = if you have specified SUPER\_BITMAP as the type of raster you want for this window, this value points to an instance of the BitMap structure. However, if the raster type is not SUPER\_BITMAP, this pointer is ignored.

MinWidth, MinHeight, MaxWidth, MaxHeight = the size limits for this window. These must be reasonable values, which is to say that the minimums cannot be greater than the current size, nor can the maximums be smaller than the current size. If they are, they are ignored. Any one of these can be initialized to zero, which means that that limit will be set to the current dimension of that axis. The limits can be changed after the window is opened by calling the WindowLimits() routine. If you have not requested the WINDOWSIZING option, these variables are ignored and you do not have to initialize them.

**RESULT**

If all is well, this command returns a pointer to the structure of your new window. If anything goes wrong, it returns NULL.

**BUGS**

ACTIVATE is currently advisory only. The user is able to do things that will prevent your window from becoming the active one when it opens.

**SEE ALSO**

OpenScreen(), ModifyIDCMP(), SetWindowTitles(), WindowLimits()

**OpenWorkBench****NAME**

OpenWorkBench -- open the Workbench screen

**SYNOPSIS**

```
BOOL OpenWorkBench()
```

**FUNCTION**

This routine attempts to reopen the Workbench. If the Workbench screen reopens successfully, this routine returns TRUE; if something goes wrong, it returns FALSE.

Even though this routine does return a BOOL value, you can ignore the return value if you want.

**INPUTS**

None

**RESULT**

TRUE if the Workbench screen opened successfully or was already opened.  
FALSE if anything went wrong and the Workbench screen is not open.

**BUGS**

None

**SEE ALSO**

None

## OrRectRegion

### NAME

OrRectRegion -- perform second OR operation of rectangle with region, leaving result in region

### SYNOPSIS

OrRectRegion(region,rectangle)  
          a0      a1

### Function

If any portion of rectangle is not in the region, adds that portion to the region

### INPUTS

region      = pointer to Region structure  
rectangle   = pointer to Rectangle structure

### BUGS

## Output

### NAME

Output -- Determine the programs initial output file handle.

### SYNOPSIS

file = Output()  
D0

### FUNCTION

To identify the program's initial output file handle, you use Output.  
(To identify the initial input, see Input.)

### RESULTS

file - BCPL pointer to a file handle

## OwnBlitter

### NAME

OwnBlitter -- get the blitter for private usage

### SYNOPSIS

OwnBlitter()

### FUNCTION

Returns when the blitter has been locked from others using it and can now be used by this task. Before actually using, the new owner should call WaitBlit, which waits until any previous blit that the blitter may have been doing is actually done.

### INPUTS

### RETURNS

### SEE ALSO

DisownBlitter

## ParentDir

### NAME

ParentDir -- obtain the parent of a directory or file

### SYNOPSIS

Lock = ParentDir( lock )  
D0                    D1

### FUNCTION

This function returns a lock associated with the parent directory of a file or directory. That is, ParentDir takes a lock associated with a file or directory and returns the lock of its parent directory.

Note: The result of ParentDir may be zero (0) for the root of the current filing system.

### INPUTS

lock - BCPL pointer to a lock

### RESULTS

lock - BCPL pointer to a lock



## PeekCLMark

### NAME

PeekCLMark -- peek at the byte in the clist at the mark

### SYNOPSIS

```
byte = PeekCLMark(cList)
D0 A0
```

### FUNCTION

Returns the byte value at the mark in the character list associated with the mark.

### INPUTS

cList - a longword descriptor for a clist that can be used for clist functions.

### RESULTS

byte - the byte at the mark in the clist.

## Permit

### NAME

Permit -- Permit multi-tasking following a Forbid()

### SYNOPSIS

```
Permit();
```

### FUNCTION

Task switching will not necessarily be permitted after this call since the Forbid() function nests (only an equal number of Permit's following a set of Forbid's finally allows task-switching).

### SEE ALSO

Forbid

## PolyDraw

### NAME

PolyDraw -- draw lines from table of (x,y) values.

### SYNOPSIS

```
PolyDraw(rp, count , array)
 a1 d0 a0
```

### FUNCTION

Starting with the first pair, draws connected lines to it and to every succeeding pair.

### INPUTS

rp = pointer to RastPort structure  
count = number of points in array (x,y) pairs  
array = pointer to first (x,y) pair

### BUGS

none known

### SEE ALSO

Draw()

## PrintIText

### NAME

PrintIText -- print the text according to the IntuiText argument

### SYNOPSIS

```
PrintIText(RastPort, IText, LeftEdge, TopEdge)
 A0 A1 D0 D1
```

### FUNCTION

This routine prints the IntuiText into the specified RastPort. It sets up the RastPort as specified by the IntuiText values, then prints the text into the RastPort at the IntuiText x,y coordinates offset by the left/top arguments.

This routine does Intuition window-clipping as appropriate. If you print text outside of your window, your characters will be clipped at the window's edge.

If the NextText field of the IntuiText argument is non-zero, the next IntuiText is drawn as well (return to the top of this FUNCTION section for details).

### INPUTS

RastPort = pointer to the RastPort destination of the text.  
IText = pointer to an IntuiText structure.  
LeftEdge = left offset of the IntuiText into the RastPort.  
TopEdge = top offset of the IntuiText into the RastPort.

### RESULT

None

### BUGS

None

### SEE ALSO

None

## PutCLBuf

NAME PutCLBuf -- convert contiguous data into a character list

SYNOPSIS  
error = PutCLBuf(cList, buffer, length)  
D0                   A0    A1    D1

FUNCTION  
Appends the contents of the data buffer to a character list.  
The buffer data remains intact.

INPUTS  
clist - The clist descriptor used to manage this character list, as returned by AllocCList.  
buffer - A pointer to byte data used to initialize the character list.  
length - The number of bytes of data in the buffer.

RESULTS  
error - non-zero indicates the number of bytes not added.

## PutCLChar

NAME PutCLChar -- add a byte to the end of a character list

SYNOPSIS  
error = PutCLChar(cList, byte)  
D0                   A0    D0

FUNCTION  
Adds a byte to the end of the character list described by the cList.

INPUTS  
clist - The clist header used to manage this character list, as returned by AllocCList or StrToCL.  
byte - The byte to add to the end of the character list

RESULTS  
error - non-zero indicates the byte could not be added

## PutCLWord

### NAME

PutCLWord -- add a word to the end of a character list

### SYNOPSIS

```
error = PutCLWord(cList, word)
D0 A0 D0
```

### FUNCTION

Add a word to the end of the character list described by the cList.

### INPUTS

cList - The cList header used to manage this character list, as returned by AllocCList or StrToCL.  
word - The word to add to the end of the character list

### RESULTS

error - non-zero indicates the number of bytes not added. Partial words are not added, so error is always zero or two.

## PutDiskObject

### NAME

PutDiskObject -- write out a DiskObject to disk

### SYNOPSIS

```
status = PutDiskObject(name, diskobj)
D0 A0 A1
```

### FUNCTION

This routine writes out a DiskObject structure and its associated information. The file name of the info file will be the name parameter with a ".info" postpended to it. If the call fails, a zero will be returned. The reason for the failure may be obtained via IoErr().

PutDiskObject and PutIcon are functionally identical. They are both provided so there is a Put/Get/Free triple for disk objects.

### INPUTS

name -- name of the object  
diskobj -- a pointer to a DiskObject

### RESULTS

status -- non-zero if the call succeeded

### EXCEPTIONS

### SEE ALSO

GetDiskObject, FreeDiskObject, PutIcon

### BUGS

## PutIcon

### NAME

PutIcon -- write out a DiskObject to disk

### SYNOPSIS

```
status = PutIcon(name, icon)
D0 A0 A1
```

### FUNCTION

This routine writes out a DiskObject structure and its associated information. The file name of the info file will be the name parameter with a ".info" postpended to it. If the call fails, a zero will be returned. The reason for the failure may be obtained via IoErr().

PutDiskObject and PutIcon are functionally identical. They are both provided so there is a Put/Get/Free triple for disk objects.

Users are encouraged to use PutDiskObject instead of this routine

### INPUTS

name -- name of the object  
icon -- a pointer to a DiskObject

### RESULTS

status -- non-zero if the call succeeded

### EXCEPTIONS

### SEE ALSO

### BUGS

## PutMsg

### NAME

PutMsg -- put a message to a message port

### SYNOPSIS

```
PutMsg(port, message)
A0 A1
```

### FUNCTION

This function attaches a message to a given message port. It provides a fast, non-copying message sending mechanism.

Messages can be attached to only one port at a time. The message body can be of any size or form. Because messages are not copied, cooperating tasks share the same message memory. The sender task should not recycle the message until it has been replied by the receiver. Of course this depends on the message handling conventions setup by the involved tasks. If the ReplyPort field is non-zero, when the message is replied by the receiver, it will be sent back to that port.

Any one of the following actions can be set to occur when a message is put:

1. no special action
2. signal a given task
3. cause a software interrupt

The action is selected depending on the value set in PB\_ACTION of MP\_FLAGS.

### INPUT

port - pointer to a message port  
message - pointer to a message

### SEE ALSO

GetMsg, ReplyMsg

## PutWBOobject

### NAME

PutWBOobject -- write out a Workbench object

### SYNOPSIS

```
status = PutWBOobject(name, object)
D0 A0 A1
```

### FUNCTION

This routine writes a Workbench object to disk. The name parameter will have a ".info" postpended to it, and that file name will have the disk-resident information written into it. If the call fails, it will return a zero. The reason for the failure may be obtained via IoErr().

This routine is intended only for internal users that can track changes to the Workbench.

### INPUTS

name -- name of the object  
object -- the Workbench object to be written out

### RESULTS

status -- non-zero if the call succeeded.

### EXCEPTIONS

### SEE ALSO

### BUGS

## QBlit

### NAME

QBlit -- queue up a request for blitter usage

### SYNOPSIS

```
QBlit(bp)
 A1
```

### FUNCTION

Links a request for the use of the blitter to the end of the current blitter queue. The pointer bp points to a blit structure containing, among other things, the link information and the address of your routine which is to be called when the blitter queue finally gets around to this specific request. When your routine is called, you are in control of the blitter ... it is not busy with anyone else's requests. This means that you can directly specify the register contents and start the blitter. See the description of the blit structure and the uses of QBlit in the "Graphics Primitives" chapter in this manual. The header of a blitter structure is shown in hardware/blit.h

### INPUTS

bp = pointer to a blit structure

### RESULT

Your routine is called when the blitter is ready for you.

### NOTE

In general, requests for blitter usage through this channel are put in front of those who use the blitter via OwnBlitter and DisownBlitter. However, for small blits there is more overhead using the queuer than Own/Disown Blitter.

### BUGS

None known

### SEE ALSO

QBSblit blit.h

## QBSblit

### NAME

QBSblit -- synchronize the blitter request with the video beam.

### SYNOPSIS

```
QBSblit(bsp)
 al
```

### FUNCTION

Calls a user routine for use of the blitter, enqueued separately from the QBlit queue. Calls the user routine contained in the blit structure when the video beam is located at a specified position onscreen. Useful when you are trying to blit into a visible part of the screen and wish to perform the data move while the beam is not trying to display that same area (prevents showing part of an old display and part of a new display simultaneously). Blitter requests on the QBSblit queue take precedence over those on the regular blitter queue. The beam position is specified through the blitnode.

### INPUTS

bsp = pointer to a blit structure. See description in the Graphics Support section of the manual for more info.

### RESULT

User routine is called when the QBSblit queue reaches this request AND the video beam is in the specified position.

### BUGS

### SEE ALSO

QBlit

## Read

### NAME

Read -- read bytes of data from a file

### SYNOPSIS

```
actualLength = Read(file, buffer, length)
 D0 D1 D2 D3
```

### FUNCTION

You can copy data with a combination of Read and Write. Read reads bytes of information from an opened file (represented here by the argument 'file') into the memory buffer indicated. Read attempts to read as many bytes as fit into the buffer as indicated by the value of length. You should always make sure that the value you give as the length really does represent the size of the buffer. Read may return a result indicating that it read less bytes than you requested, for example, when reading a line of data that you typed at the terminal.

The value returned is the length of the information actually read. That is to say, when 'actualLength' is greater than zero, the value of 'actualLength' is the the number of characters read. A value of zero means that end-of-file has been reached. Errors are indicated by a value of -1. Read from the console returns a value when a return is found or the buffer is full.

A call to Read also modifies or changes the value of IoErr(). IoErr() gives more information about an error (for example, actualLength equals -1) when it is called.

### INPUTS

file - BCPL pointer to a file handle  
buffer - address of the first location of a buffer  
length - integer

### RESULTS

actualLength - integer

## ReadPixel

### NAME

ReadPixel -- read the pen number value of the pixel at a specified x,y location within a certain RastPort

### SYNOPSIS

```
penno = (int)ReadPixel(rp, x, y)
 D0 al D0 D1
```

### FUNCTION

Combines the bits from each of the bit-planes used to describe a particular RastPort into the pen number selector which that bit combination normally forms for the system hardware selection of pixel color.

### INPUTS

x is the X coordinate within the range of the RastPort size.  
y is the Y coordinate within the range of the RastPort size.  
rp is a pointer to a RastPort structure  
rp is a pointer to a RastPort structure

### RESULT

Pen (0..255) number at that position is returned.  
-1 is returned if cannot read that pixel

### BUGS

### SEE ALSO

WritePixel

## RectFill

### NAME

RectFill -- fill a defined rectangular area with the current drawing pen color, outline color, secondary color, and pattern.

### SYNOPSIS

```
RectFill(rp, xmin, ymin, xmax, ymax)
 A1 D0 D1 D2 D3
```

### FUNCTION

Fills the rectangular region specified by the parameters with the chosen pen colors, areafill pattern, and drawing mode.

### INPUTS

(xmin,ymin) (xmax,ymax) are the coordinates of the upper left corner and the lower right corner, respectively, of the rectangle.  
(xmax >= xmin) and (ymax >= ymin)

rp points to the RastPort which receives the filled rectangle.

### SEE ALSO



## RefreshGadgets

### NAME

RefreshGadgets -- refresh (redraws) the gadget display

### SYNOPSIS

```
RefreshGadgets(Gadgets, Pointer, Requester)
 A0 A1 A2
```

### FUNCTION

This routine refreshes (redraws) all of the gadgets in the gadget list, starting from the specified gadget.

The Pointer argument points to a Window structure.

The Requester variable can point to a Requester structure. If the first gadget in the list has the REQGADGET flag set, the gadget list refers to gadgets in a requester and Pointer must necessarily point to a window. If these are not the gadgets of a requester, the Requester argument may be NULL.

There are two main reasons why you might want to use this routine. First, you have modified the imagery of the gadgets in your display and you want the new imagery to be displayed. Second, if you think that some graphic operation trashed the gadgetry of your display, this routine will refresh the imagery.

The Gadgets argument can be a copy of the FirstGadget variable in either the Screen or Window structure that you want refreshed; the effect of this will be that all gadgets will be redrawn. However, you can selectively refresh just some of the gadgets by starting the refresh part way into the list—for instance, redrawing your window non-GIMMEZEROZERO gadgets only, which you have conveniently grouped at the end of your gadget list.

NOTE: It is never safe to tinker with the gadget list yourself. Do not supply some gadget list that Intuition has not already processed in the usual way.

NOTE: If you have specified that this is the gadget list of a requester, that requester must be currently displayed.

### INPUTS

Gadgets = pointer to the first structure in the list of gadgets wanting refreshment.

Pointer = pointer to a Window structure.

Requester = pointer to a Requester structure (may be NULL if this is not a requester gadget list).

### RESULT

None

### BUGS

None

### SEE ALSO

None

## RemakeDisplay

### NAME

RemakeDisplay -- remake the entire Intuition display

### SYNOPSIS

```
RemakeDisplay()
```

### FUNCTION

This is the big one. This procedure remakes the entire Intuition display. It calls MakeScreen() for every screen in the system and then it calls RethinkDisplay(), which rethinks the relationships of the screens to one another and then rethinks the display Copper lists.

WARNING: This routine can take several milliseconds to run, so do not use it lightly. RethinkDisplay() (called by this routine) does a Forbid() on entry and a Permit() on exit, which can seriously degrade the performance of the multitasking Executive.

### INPUTS

None

### RESULT

None

### BUGS

None

### SEE ALSO

RethinkDisplay()

## RemDevice

### NAME

RemDevice -- remove a device from the system

### SYNOPSIS

```
error = RemDevice(device)
D0 A1
```

### FUNCTION

This function removes an existing device from the system.  
This function deletes the device from the device name list,  
so no new opens can occur.

### INPUTS

device - pointer to a device node

### RESULTS

error - zero if successful, else an error is returned

### SEE ALSO

AddDevice

## RemFont

### NAME

RemFont -- remove a font from the system list

### SYNOPSIS

```
error = RemFont(textFont), GraphicsLib
D0 A1 A6
```

### FUNCTION

This function removes a font from the system, ensuring that  
access to it is restricted to those applications that  
currently have an active pointer to it: i.e., no new GetFont  
requests to this font are satisfied.

### INPUTS

textFont - the TextFont structure to remove.

RemHead

NAME RemHead -- remove the head node from a list

SYNOPSIS  
node = RemHead(list)  
D0 A0

FUNCTION  
Get a pointer to the head node and remove it from the list.

INPUTS  
list - a pointer to the target list header

RESULT  
node - the node removed or zero when empty list

RemIBob

NAME RemIBob -- immediately remove a Bob from the GEL list and the RastPort

SYNOPSIS  
RemIBob(Bob, RPort, VPort)  
a0 a1 a2

FUNCTION  
Removes a Bob immediately by uncoupling it from the GEL list and erasing it from the RastPort

INPUTS  
Bob = pointer to the Bob to be removed  
RPort = pointer to the RastPort if the Bob is to be erased  
VPort = pointer to the ViewPort for beam-synchronizing

RESULT  
Nothing

BUGS  
None known

SEE ALSO  
RemVSprite

## RemIntServer

### NAME

RemIntServer -- remove an interrupt server

### SYNOPSIS

```
RemIntServer(intNum, interrupt)
D0=0:4 A1
```

### FUNCTION

This function removes an interrupt server node from the given server chain.

If this server was the last one in the chain interrupts will be disabled for intNum.

### INPUTS

intNum - the Paula interrupt bit (0..14)  
interrupt - pointer to an interrupt server node

### SEE ALSO

AddIntServer

## RemLibrary

### NAME

RemLibrary -- remove a library from the system

### SYNOPSIS

```
error = RemLibrary(library)
D0 A1
```

### FUNCTION

This function removes an existing library from the system. It will delete it from the system library name list, so no new opens may be performed.

### INPUTS

library - pointer to a library node structure

### RESULTS

error - zero if successful, else an error number

### SEE ALSO

AddLibrary

## Remove

### NAME

Remove -- remove a node from a list

### SYNOPSIS

Remove(node)  
A1

### FUNCTION

Remove a node from a list.

### INPUTS

node - the node to remove

## RemoveGadget

### NAME

RemoveGadget -- remove a gadget from a window

### SYNOPSIS

USHORT RemoveGadget(Pointer, Gadget)  
A0 A1

### FUNCTION

This routine removes the given gadget from the gadget list of the specified window. It returns the ordinal position of the removed gadget. If the gadget pointer points to a gadget that is not in the appropriate list, -1 is returned. If there are no gadgets in the list, -1 is returned. If you remove the 65,535th gadget from the list, -1 is returned.

NOTE: The gadget's imagery is not erased by this routine.

### INPUTS

Pointer = pointer to the window from which the gadget is to be removed.

Gadget = pointer to the gadget to be removed. The gadget itself describes whether this gadget should be removed from the window.

### RESULT

Returns the ordinal position of the removed gadget. If the gadget was not found in the appropriate list or if there are no gadgets in the list, -1 is returned.

### BUGS

None

### SEE ALSO

AddGadget()

### RemPort

### NAME

RemPort -- remove a message port from the system

### SYNOPSIS

RemPort(port)  
A1

### FUNCTION

This function removes a message port structure from the system's message port list. Subsequent attempts to rendezvous by name with this port will fail.

### INPUTS

port - pointer to a message port

### SEE ALSO

AddPort, FindPort

## RemResource

### NAME

RemResource -- remove a resource from the system

### SYNOPSIS

```
RemResource(resource)
 A1
```

### FUNCTION

This function removes an existing resource from the system.

### INPUTS

resource - pointer to a resource node

### SEE ALSO

AddResource

## RemTail

### NAME

RemTail -- remove the tail node from a list

### SYNOPSIS

```
node = RemTail(list)
 D0 A0
```

### FUNCTION

Get a pointer to the tail node and remove it from the list.

### INPUTS

list - a pointer to the target list header

### RESULT

node - the node removed or zero when empty list

## RemTask

### NAME

RemTask -- remove a task from the system

### SYNOPSIS

```
RemTask(task)
 A1
```

### FUNCTION

This function removes a task from the system. Deallocation of resources should have been performed prior to calling this function.

### INPUTS

task - pointer to the task node representing the task to be removed. A zero value indicates self removal, and will cause the next ready task to begin execution.

### SEE ALSO

AddTask

## RemVSprite

### NAME

RemVSprite -- remove a VSprite from the current GEL list

### SYNOPSIS

```
RemVSprite(VS)
 a0
```

### FUNCTION

Unlinks the VSprite from the current GEL list

### INPUTS

VS = pointer to the VSprite structure to be removed from the GEL list

### RESULT

Nothing

### BUGS

None known

### SEE ALSO

Nothing

## Rename

### NAME

Rename -- rename a directory or file

### SYNOPSIS

```
success = Rename(oldName, newName)
 D0 D1 D2
```

### FUNCTION

Rename attempts to rename the file or directory specified as 'oldName' with the name 'newName'. If the file or directory 'newName' exists, Rename fails and Rename returns an error.

Both the 'oldName' and the 'newName' can be complex filenames containing a directory specification. In this case, the file will be moved from one directory to another. However, the destination directory must exist before you do this.

Note: It is impossible to rename a file from one volume to another.

### INPUTS

oldName - address of first character of a null-terminated string  
newName - address of first character of a null-terminated string

### RESULTS

success - boolean

## ReplyMsg

### NAME

ReplyMsg -- put a message to its reply port

### SYNOPSIS

```
ReplyMsg(message)
 A1
```

### FUNCTION

This function sends a message to its reply port. This is usually done when the receiver of a message has finished and wants to return it to the sender (so that it can be re-used or deallocated, whatever).

### INPUT

message - a pointer to the message

### SEE ALSO

ReplyMsg



## ReportMouse

### NAME

ReportMouse -- tell Intuition whether or not to report mouse movement

### SYNOPSIS

ReportMouse(Window, Boolean)  
A0 D0

### FUNCTION

This routine tells Intuition whether or not to broadcast mouse movement events to this window when it is active. The Boolean value specifies whether to start or stop broadcasting position information of mouse-movement. If the window is active, mouse-movement reports start coming immediately after this command. This routine will change the current state of the FOLLOWMOUSE function of a currently-selected gadget, too. Note that calling ReportMouse() when a gadget is selected will only temporarily change whether or not mouse movements are reported while the gadget is selected; the next time the gadget is selected, its FOLLOWMOUSE flag is examined anew. Note also that calling ReportMouse() when no gadget is currently selected will change the state of the window's REPORTMOUSE flag but will have no effect on any gadget that may be subsequently selected.

The ReportMouse() function is first performed when OpenWindow() is first called. If the flag REPORTMOUSE is included among the options, all mouse-movement events are reported to the opening task and will continue to be reported until ReportMouse() is called with a Boolean value of FALSE. If REPORTMOUSE is not set, no mouse-movement reports will be broadcast until ReportMouse() is called with a Boolean value of TRUE.

### INPUTS

Window = pointer to a Window structure associated with this request.  
Boolean = TRUE or FALSE value specifying whether to turn this function on or off.

### RESULT

None

### BUGS

None

### SEE ALSO

None

## Request

### NAME

Request -- activate a requester

### SYNOPSIS

Request(Requester, Window)  
A0 A1

### FUNCTION

This routine links in and displays a requester in the specified window. This routine ignores the window's REQVERIFY flag.

### INPUTS

Requester = pointer to the structure of the requester to be displayed.  
Window = pointer to the structure of the window into which this requester goes.

### RESULT

If the requester is successfully opened, TRUE is returned.  
If the requester could not be opened, FALSE is returned.

### BUGS

None

### SEE ALSO

None

## RethinkDisplay

### NAME

RethinkDisplay -- the grand manipulator of the entire Intuition display

### SYNOPSIS

RethinkDisplay()

### FUNCTION

This function performs the Intuition global display reconstruction. This includes massaging internal-state data, rethinking all of the ViewPorts and their relationship to one another, and, finally, reconstructing the entire display based on the results of all this rethinking.

The reconstruction of the display includes calls to the graphics library to perform MrgCop() and LoadView() for all of Intuition's screens.

You may perform a MakeScreen() on your custom screen before calling this routine. The results will be incorporated in the new display.

**WARNING:** This routine can take several milliseconds to run, so do not use it lightly. RethinkDisplay() does a Forbid() on entry and a Permit() on exit, which can seriously degrade the performance of the multitasking Executive.

### INPUTS

None

### RESULT

None

### BUGS

None

### SEE ALSO

MakeScreen(), RemakeDisplay(), MrgCop(), LoadView(), Forbid(), Permit()

## ScrollLayer

### NAME

ScrollLayer -- scroll around in a superbitmap

### SYNOPSIS

ScrollLayer( li, l, dx, dy )  
                  a0 al d0 d1

### INPUTS

li = pointer to LayerInfo structure  
l = pointer to a nonbackdrop layer  
dx = delta to add to current x scroll value  
dy = delta to add to current y scroll value

### FUNCTION

Copies bits between layer and superbitmap to reposition layer over different portion of superbitmap.

### SEE ALSO

layers.h

## ScreenToBack

NAME  
ScreenToBack -- send the specified screen to the back of the display

SYNOPSIS  
ScreenToBack(Screen)  
A0

FUNCTION  
This routine sends the specified screen to the back of the display.

INPUTS  
Screen = pointer to a Screen structure

RESULT  
None

BUGS  
None

SEE ALSO  
ScreenToFront()

## ScreenToFront

NAME  
ScreenToFront -- bring the specified screen to the front of the display

SYNOPSIS  
ScreenToFront(Screen)  
A0

FUNCTION  
This routine brings the specified screen to the front of the display.

INPUTS  
Screen = a pointer to a Screen structure

RESULT  
None

BUGS  
None

SEE ALSO  
ScreenToBack()

## ScrollRaster

### NAME

ScrollRaster -- push bits in rectangle in raster around by dx,dy towards 0,0 inside rectangle

### SYNOPSIS

```
ScrollRaster(rp, dx, dy, xmin, ymin, xmax, ymax)
 a1 d0 d1 d2 d3 d4 d5
```

### FUNCTION

Moves the bits in the raster by (dx,dy) towards (0,0). The space vacated is RectFilled with BGPen. Limits the scroll operation to the rectangle defined by (xmin,ymin)(xmax,ymax). Bits outside will not be affected.

### INPUTS

rp must be a valid pointer to a RastPort  
dx,dy are integers that may be positive, zero, or negative

### EXAMPLE

```
ScrollRaster(rp,0,1) /* shift raster up by one row */
ScrollRaster(rp,-1,-1) /* shift raster down and to the right by 1 pixel
```

### BUGS

## ScrollVPort

### NAME

ScrollVPort -- push bits in rectangle in vport around by dx,dy towards 0,0 inside rectangle

### SYNOPSIS

```
ScrollVPort(vp)
 a0
```

### FUNCTION

After the programmer has adjusted the Offset values in the RasInfo structures of ViewPort, changes the the Copper lists to reflect the the scroll positions.

### INPUTS

vp must be a valid pointer to a ViewPort that is currently on display

### RESULTS

Modifies hardware and intermediate Copper lists to reflect new RasInfo

### NOTE

Changing the BitMap ptr in RasInfo and not changing the the Offsets will cause a double-buffering affect.

### BUGS

Pokes not fast enough to avoid some visible hashing of display

## Seek

## NAME

Seek -- move to a logical position in a file

## SYNOPSIS

```
oldPosition = Seek(file, position, mode)
 D0 D1 D2 D3
```

## FUNCTION

Seek sets the read/write cursor for the file 'file' to the position 'position'. Both Read and Write use this position as a place to start reading or writing. If all goes well, the result is the previous position

in the file. If an error occurs, the result is -1. You can then use IoErr() to find out more information about the error.

'mode' can be OFFSET\_BEGINNING (=-1), OFFSET\_CURRENT (=0) or OFFSET\_END (=1). You use it to specify the relative start position. For example, 20 from current is a position twenty bytes forward from current, -20 from end is 20 bytes before the end of the current file.

To find out the current file position without altering it, you call to Seek specifying an offset of zero from the current position.

To move to the end of a file, Seek to end-of-file offset with zero position. Note that you can append information to a file by moving to the end of a file with Seek and then writing. You cannot Seek beyond the end of a file.

## INPUTS

file - BCPL pointer to a file handle  
position - integer  
mode - integer

## RESULTS

oldPosition - integer

## SendIO

## NAME

SendIO -- initiate an I/O command

## SYNOPSIS

```
SendIO(iORequest)
 A1
```

## FUNCTION

This function requests the device driver to initiate the command specified in the given I/O request. The device will return regardless of whether the I/O has completed.

## INPUTS

iORequest - pointer to an I/O request

## SEE ALSO

DoIO, WaitIO

SetAPen

NAME

SetAPen -- Set primary pen

SYNOPSIS

```
SetAPen(rp, pen)
 al d0
```

FUNCTION

Sets the primary drawing pen for lines, fills, and text.

INPUTS

rp = pointer to RastPort structure.  
pen = 0-255

RESULT

Changes the minterms in the RastPort to reflect new primary pen.  
Set line drawer to restart pattern.

BUGS

SEE ALSO

SetBPen

SetBPen

NAME

SetBPen -- Set secondary pen

SYNOPSIS

```
SetBPen(rp, pen)
 al d0
```

FUNCTION

Sets the secondary drawing pen for lines, fills, and text.

INPUTS

rp = pointer to RastPort structure.  
pen = 0-255

RESULT

Changes the minterms in the RastPort to reflect new secondary pen.  
Set line drawer to restart pattern.

BUGS

SEE ALSO

SetAPen

## SetCollision

### NAME

SetCollision -- sets a pointer to a user collision routine

### SYNOPSIS

```
SetCollision(num, routine, GInfo)
 d0 a0 al
```

### FUNCTION

Sets entry h in the user's collision vectors table equal to the pointer p

### INPUTS

num = collision vector number  
routine = pointer to the user's collision routine  
GInfo = pointer to a GelsInfo structure

### RESULT

Nothing

### BUGS

None known

### SEE ALSO

Nothing

## SetComment

### NAME

SetComment -- set a comment

### SYNOPSIS

```
Success = SetComment(name, comment)
 D0 D1 D2
```

### FUNCTION

SetComment sets a comment on a file or directory. The comment is a pointer to a null-terminated string of up to 80 characters.

### INPUTS

name - address of first character of a null-terminated string  
comment - address of first character of a null-terminated string

### RESULTS

success - boolean

## SetDMRequest

### NAME

SetDMRequest -- set the DMRequest of the window

### SYNOPSIS

```
SetDMRequest(Window, DMRequester)
 A0 A1
```

### FUNCTION

This routine attempts to set the DMRequester in the specified window. The DMRequester is the special requester that you attach to the double-click of the menu button, allowing the user to bring up this requester on demand. This routine will not set the DMRequester if it is already set and is currently active (in use by the user). To change the DMRequester after having called SetDMRequest(), you start by calling ClearDMRequest() until it returns a value of TRUE. Then you can call SetDMRequest() with the new DMRequester.

### INPUTS

Window = pointer to the structure of the window into which the DMRequest is to be set.  
DMRequester = a pointer to a Requester structure.

### RESULT

If the current DMRequest was not in use, the DMRequester pointer is set in the window and this routine returns TRUE.

If the DMRequest was currently in use, this routine does not change the pointer and returns FALSE.

### BUGS

None

### SEE ALSO

ClearDMRequest(), Requester().

## SetDrMd

### NAME

SetDrMd -- set drawing mode

### SYNOPSIS

```
SetDrMd(rp, mode)
 al d0
```

### FUNCTION

Sets the drawing mode for lines, fills and text.

### INPUTS

rp = pointer to RastPort structure.  
mode = 0-255

```
#define JAM1 0 /* jam 1 color into raster */
#define JAM2 1 /* jam 2 colors into raster */
#define COMPLEMENT 2 /* XOR bits into raster */
#define INVERSVID 4 /* inverse video for drawing modes */
```

Some combinations may not make much sense.

### RESULT

The mode set is dependent on the bits selected.  
Change minterms to reflect new drawing mode.  
Set line drawerto restart pattern.

### BUGS

### SEE ALSO

SetAPen



## SetExcept

### NAME

SetExcept -- define certain signals to cause exceptions

### SYNOPSIS

```
oldSignals = SetExcept(newSignals, signalMask)
D0 D0 D1
```

### FUNCTION

This function defines which of the task's signals will cause an exception. When any of the signals occurs the task's exception handler will be dispatched. If the signal occurred prior to calling SetExcept, the exception will happen immediately.

### INPUTS

newSignals - the new values for the signals specified in signalMask.  
signalMask - the set of signals to be effected

### RESULTS

oldSignals - the prior exception signals

### EXAMPLE

```
Get the current state of all exception signals:
SetExcept(0,0)
Change a few exception signals:
SetExcept($1374,$1074)
```

### SEE ALSO

Signal, SetSignal

## SetFont

### NAME

SetFont -- set the text font and attributes in a RastPort

### SYNOPSIS

```
error = SetFont(rastPort, font), graphicsLib
D0 A1 A0 A6
```

### FUNCTION

This function sets the font in the RastPort to that described by font and updates the text attributes to reflect that change. If TextAttr is zero, this call leaves the RastPort with no font. This function clears the effect of any previous soft styles.

### INPUTS

RastPort - the RastPort in which the text attributes are changed.  
font - an open font.

## SetFunction

### NAME

SetFunction -- change a function vector in a library

### SYNOPSIS

```
oldFunc = SetFunction(library, funcOffset, funcEntry)
D0 Al AO.W D0
```

### FUNCTION

SetFunction is a functional way of changing those parts of a library that are checksummed. They are changed in such a way that the summing process will never falsely declare a library to be invalid.

### INPUTS

library - a pointer to the library to be changed

funcOffset - the offset that FuncEntry should be put at.

funcEntry - pointer to new function

## SetIntVector

### NAME

SetIntVector -- set a system interrupt vector

### SYNOPSIS

```
oldInterrupt = SetIntVector(intNumber, interrupt)
D0 D0-0:4 Al
```

### FUNCTION

This function provides a mechanism for setting the system interrupt vectors. Both the code and data pointers of the vector are set to the new values. A pointer to the old interrupt structure is returned. When the system calls the specified interrupt code the registers are setup as follows:

D0 - scratch

D1 - scratch (on entry: active portia interrupts)

A0 - scratch (on entry: pointer to chipbase)

Al - scratch (on entry: interrupt's data segment)

A5 - jump vector register (scratch on call)

A6 - library base pointer (scratch on call)

all other registers - must be preserved

### INPUTS

intNum - the Paula interrupt bit number (0..14)

interrupt - a pointer to a node structure containing the handler's entry point and data segment pointer. It is a good idea to give the node a name so that other users may identify who currently has control of the interrupt.

### RESULT

A pointer to the prior interrupt node which had control of this interrupt.

## SetMenuStrip

### NAME

SetMenuStrip -- attach the menu strip to the window

### SYNOPSIS

```
SetMenuStrip(Window, Menu)
 A0 A1
```

### FUNCTION

This routine attaches the menu strip to the window. If the user presses the menu button after this routine is called, this specified menu strip will be displayed and accessible.

NOTE: You should always design your menu strip changes to be two-way operations; every menu strip you add to your window should be cleared sometime. Even in the simplest case, when you will have just one menu strip for the lifetime of your window, you should always clear the menu strip before closing the window. If you already have a menu strip attached to this window, the correct procedure for changing to a new menu strip involves calling ClearMenuStrip() to clear the old menu strip first. The sequence of events should be:

1. OpenWindow().
2. Zero or more iterations of:
  - o SetMenuStrip().
  - o ClearMenuStrip().
3. CloseWindow().

### INPUTS

Window = pointer to a Window structure.  
Menu = pointer to the first Menu structure in the menu strip.

### RESULT

None

### BUGS

None

### SEE ALSO

ClearMenuStrip()

## SetOPen

### NAME

SetOPen -- Set outline pen

### SYNOPSIS

```
SetOPen(rp, pen)
 al d0
```

### FUNCTION

Set the outline drawing pen for area outlines.

### INPUTS

rp = pointer to RastPort structure.  
pen = 0-255

### RESULT

Changes the minterms in the RastPort to reflect new outline pen.

### BUGS

### SEE ALSO

## SetPointer

### NAME

SetPointer -- set a window with its own pointer

### SYNOPSIS

```
SetPointer(Window, Pointer, Height, Width, Xoffset, Yoffset)
 A0 A1 D0 D1 D2 D3
```

### FUNCTION

This routine sets up the window with the sprite definition for the pointer. Then, whenever the window is active, the pointer image will change to the sprite's version of the pointer. If the window is active when this routine is called, the change takes place immediately.

The Xoffset and Yoffset arguments are used to offset the top left corner of the hardware sprite imagery from what Intuition regards as the current position of the pointer.

Another way of describing it is as the offset from the "hot spot" of the pointer to the top left corner of the sprite. For instance, if you specify offsets of zero, zero, then the top-left corner of your sprite image will be placed at the pointer position. On the other hand, if you specify an Xoffset of -7 (remember, sprites are 16 pixels wide), your sprite will be centered over the pointer position. If you specify an Xoffset of -15, the right edge of the sprite will be over the pointer position.

### INPUTS

Window = pointer to the structure of the window to receive this pointer definition.  
Pointer = pointer to the data definition of a sprite.  
Height = the height of the pointer.  
Width = the width of the sprite (must be less than or equal to 16).  
Xoffset = the offset for your sprite from the pointer position.  
Yoffset = the offset for your sprite from the pointer position.

### RESULT

None

### BUGS

None

### SEE ALSO

ClearPointer()

## SetProtection

### NAME

SetProtection -- set file or directory protection

### SYNOPSIS

```
Success = SetProtection(name, mask)
 D0 D1 D2
```

### INPUTS

name - address of first character of a null-terminated string  
mask - the protection mask required

### RESULTS

success - boolean

### FUNCTION

SetProtection sets the protection attributes on a file or directory. The lower four bits of the mask are as follows:

bit 3: if 1 then reads not allowed, else reads allowed.  
bit 2: if 1 then writes not allowed, else writes allowed.  
bit 1: if 1 then execution not allowed, else execution allowed.  
bit 0: if 1 then deletion not allowed, else deletion allowed.

Bits 31-4 Reserved.

Only delete is checked for in the current release of AmigaDOS. Rather than referring to bits by number you should use the definitions in "include/libraries/dos.h."

## SetRast

### NAME

SetRast -- set an entire drawing area to a specified color

### SYNOPSIS

```
SetRast(RastPort, pen)
 A1 D0
```

### FUNCTION

Sets the entire contents of the specified RastPort to the specified pen.

### INPUTS

RastPort is a pointer to the rastPort you wish to use.  
Pen is the pen value which you wish to fill into that port. (0-255)

### RESULT

The drawing area becomes the selected pen number.

### BUGS

### SEE ALSO

## SetRGB4

### NAME

SetRGB4 -- set one color register for this viewport

### SYNOPSIS

```
SetRGB4(vp, n, r, g, b)
 a0 D0 D1 D2 D3
```

### INPUTS

vp= ViewPort to affect  
n = the color number (range from 0 to 31)

r = red level  
g = green level  
b = blue level

### RESULT

If there is a ColorMap for this ViewPort, store the value in the structure ColorMap.  
The selected color register is changed to match your specs.  
If the color value is unused, nothing will happen.

### BUGS

If the color value is unused it may affect the color values in the next ViewPorts.

### SEE ALSO

LoadRGB4

## SetSignal

### NAME

SetSignal -- define the state of this task's signals

### SYNOPSIS

```
oldSignals = SetSignal(newSignals, signalMask)
D0 D0 D1
```

### FUNCTION

This function defines the states of the task's signals.

This function is considered dangerous.

### INPUTS

newSignals - the new values for the signals specified in  
signalSet.  
signalMask - the set of signals to be effected

### RESULTS

oldSignals - the prior values for all signals

### EXAMPLE

```
Get the current state of all signals:
 SetSignal(0,0)
Clear all signals:
 SetSignal(0,FFFFFFFFH)
```

### SEE ALSO

Signal, Wait

## SetSoftStyle

### NAME

SetSoftStyle -- set the soft style of the current font

### SYNOPSIS

```
newStyle = SetSoftStyle(rastPort, style, enable), graphicsLib
A1 D0 D1 A6
```

### FUNCTION

This function alters the soft style of the current font. Only those bits that are also set in enable are affected. The resulting style is returned, since some style request changes will not be honored when the implicit style of the font precludes changing them.

### INPUTS

rastPort - the RastPort from which the font and style are extracted.  
style - the new font style to set, subject to enable.  
enable - those bits in style to be changed. Any set bits here that would not be set as a result of AskSoftStyle will be ignored, and the newStyle result will not be as expected.

### RESULTS

style - the resulting style, both as a result of previous soft style selection, the effect of this function, and the style inherent in the set font.

## SetSR

### NAME

SetSR -- get and/or set processor status register

### SYNOPSIS

```
oldSR = SetSR(newSR, mask)
D0 D0 D1
```

### FUNCTION

This function provides a means of modifying the CPU status register in a "safe" way (well, how safe can a function like this be anyway?). This function will only effect the status register bits specified in the mask parameter. The prior content of the entire status register is returned.

### INPUTS

newSR - new values for bits specified in the mask.  
All other bits are not effected.  
mask - bits to be changed

### RESULTS

oldSR - the entire status register before new bits

### EXAMPLES

```
To get the current SR:
 currentSR = SetSR(0,0);
To change the processor interrupt level to 3:
 oldSR = SetSR($0300,$0700);
Set processor interrupts back to prior level:
 SetSR(oldSR,$0700);
```

## SetTaskPri

### NAME

SetTaskPri -- get and set the priority of a task

### SYNOPSIS

```
oldPriority = SetTaskPri(task, priority)
D0-0:8 A1 D0-0:8
```

### FUNCTION

This function changes the priority of a task regardless of its state. The old priority of the task is returned. A reschedule is performed, and a context switch may result.

### INPUTS

task - task to be affected  
priority - the new priority for the task

### RESULT

oldPriority - the tasks previous priority

## SetWindowTitles

### NAME

SetWindowTitles -- set the window's titles for both the window and the screen

### SYNOPSIS

```
SetWindowTitles(Window, WindowTitle, ScreenTitle)
 A0 A1 A2
```

### FUNCTION

This routine allows you to set the text that appears in the window and/or screen title bars. The window title appears at all times in the window title bar. The window's screen title appears at the screen title bar whenever this window is active.

When this routine is called, your window title will be changed immediately. If your window is active when this routine is called, the screen title will be changed immediately.

You can specify a value of -1 for either of the title pointers. This designates that you want Intuition to leave the current setting of that particular title alone, modifying only the other one. Of course, you could set both to -1.

Furthermore, you can set a value of 0 for either of the title pointers. Doing so specifies that you want no title to appear (the title bar will be blank).

### INPUTS

Window = pointer to your Window structure.  
WindowTitle = pointer to a null-terminated text string; this pointer can also be set to either -1 or 0.  
ScreenTitle = pointer to a null-terminated text string; this pointer can also be set to either -1 or 0.

### RESULT

None

### BUGS

None

### SEE ALSO

OpenWindow(), ShowTitle()

## ShowTitle

### NAME

ShowTitle -- set the screen title bar display mode

### SYNOPSIS

```
ShowTitle(Screen, ShowIt)
 A0 D0
```

### FUNCTION

This routine sets the SHOWTITLE flag of the specified screen and then coordinates the redisplay of the screen and its windows.

The screen title bar can appear either in front of or behind Backdrop windows. Non-Backdrop windows always appear in front of the screen title bar. You specify whether you want the screen title bar to be in front of or behind the screen's Backdrop windows by calling this routine.

The ShowIt argument should be set to either TRUE or FALSE. If TRUE, the screen's title bar will be shown in front of Backdrop windows. If FALSE, the title bar will be located behind all windows. When a screen is first opened, the default setting of the SHOWTITLE flag is TRUE.

### INPUTS

Screen = pointer to a Screen structure.  
ShowIt = Boolean TRUE or FALSE describing whether to show or hide the screen title bar.

### RESULT

None

### BUGS

None

### SEE ALSO

SetWindowTitles()



## Signal

### NAME

Signal -- signal a task

### SYNOPSIS

```
Signal(task, signals)
 Al D0
```

### FUNCTION

This function signals a task with the given signals. If the task is currently waiting for one or more of these signals, it will be made ready and a reschedule will occur. If the task is not waiting for any of these signals, the signals will be posted to the task for possible later use. A signal may be sent to a task regardless of whether it's running, ready, or waiting.

This function is considered "low level". Its main purpose is to support multiple higher level functions like PutMsg. Generally a user need not perform Signals directly.

### INPUT

task - the task to be signalled  
signals - the signals to be sent

### SEE ALSO

Wait, SetSignal

## SizeCList

### NAME

SizeCList -- get the number of bytes in a character list

### SYNOPSIS

```
bytes = SizeCList(cList)
 D0 A0
```

### FUNCTION

Inquires as to the number of characters in cList.

### INPUTS

cList - The cList header used to manage this character list, as returned by AllocCList or StrToCL.

### RESULTS

bytes - the number of bytes in cList.

## SizeLayer

### NAME

SizeLayer -- change the size of this nonbackdrop layer.

### SYNOPSIS

```
SizeLayer(li, l, dx, dy)
 a0 al d0 d1
```

### INPUTS

li = pointer to LayerInfo structure  
l = pointer to a nonbackdrop layer  
dx = delta to add to current x size  
dy = delta to add to current y size

### FUNCTION

Changes the size of this layer by (dx,dy). The lower right hand corner is extended to make room for the larger layer. If there is SuperBitMap for this layer, copy pixels into or out of the layer depending on whether the layer increases or decreases in size. Collect damage list for those layers that may need to be refreshed if damage occurred.

### NOTE

The current implementation forces layer to front. This is not to be depended upon and may change in future releases of layer.lib.

### SEE ALSO

layers.h

## SizeWindow

### NAME

SizeWindow -- ask Intuition to size a window

### SYNOPSIS

```
SizeWindow(Window, DeltaX, DeltaY)
 A0 D0 D1
```

### FUNCTION

This routine sends a request to Intuition asking to size the window by the specified amounts. The delta arguments describe how much to size the window along the respective axes.

Note that the window will not be sized immediately. It will be sized the next time Intuition receives an input event, which happens currently at a minimum rate of ten times per second and a maximum of sixty times a second. You can discover when your window has finally been sized by setting the NEWSIZE flag of the IDCMP of your window. See the "Input and Output Methods" chapter in "Amiga Intuition Reference Manual" for a description of the IDCMP.

This routine does no error-checking. If your delta values specify some far corner of the universe, Intuition will attempt to size your window to that far corner. Because of the distortions in the space-time continuum that can result from this, as predicted by special relativity, the result is generally not desirable.

### INPUTS

Window = pointer to the structure of the window to be sized.  
DeltaX = signed value describing how much to size the window on the x axis.  
DeltaY = signed value describing how much to size the window on the y axis.

### RESULT

None

### BUGS

None

### SEE ALSO

MoveWindow(), WindowToFront(), WindowToBack()

## SortGList

### NAME

SortGList -- sort the current GEL list according to the y,x coordinates

### SYNOPSIS

SortGList(RPort) as called by C  
al

### FUNCTION

Sorts the current GEL list according to the GEL's y,x coordinates  
This sorting is essential before calls to DrawGList or DoCollision

### INPUTS

RPort = pointer to the RastPort structure containing the GelsInfo

### RESULT

Nothing

### BUGS

None known

### SEE ALSO

DoCollision  
DrawGList

## SPAbs

### NAME

SPAbs -- obtain the absolute value of the fast floating-point number

### C USAGE

```
fnum2 = SPAbs(fnum1);
DO
```

### FUNCTION

Accepts a floating-point number and returns the absolute value of  
said number.

### INPUTS

fnum1 - floating-point number

### RESULT

fnum2 - floating-point absolute value of fnum1

### BUGS

None

### SEE ALSO

SPAcos

NAME

SPAcos -- obtain the arccosine of the floating-point number

SYNOPSIS

```
fnum2 = SPAcos(fnum1);
 D0
```

FUNCTION

Accepts a floating-point number representing the cosine of an angle and returns the value of said angle in radians

INPUTS

fnum1 - floating-point number

RESULT

fnum2 - floating-point number

BUGS

SEE ALSO

SPAdd

NAME

SPAdd -- add two floating-point numbers

C USAGE

```
fnum3 = SPADD(fnum1, fnum2);
 D1 D0
```

FUNCTION

Accepts two floating-point numbers and returns the arithmetic sum of said numbers.

INPUTS

fnum1 - floating-point number

fnum2 - floating-point number

RESULT

fnum3 - floating-point number

BUGS

None

SEE ALSO

SPAsin

NAME

SPAsin -- obtain the arcsine of the floating-point number

SYNOPSIS

```
fnum2 = SPAsin(fnum1);
 D0
```

FUNCTION

Accepts a floating-point number representing the sine of an angle and returns the value of said angle in radians

INPUTS

fnum1 - floating-point number

RESULT

fnum2 - floating-point number

BUGS

None

SEE ALSO

SPAtan

NAME

SPAtan -- obtain the arctangent of the floating-point number

SYNOPSIS

```
fnum2 = SPAtan(fnum1);
 D0
```

FUNCTION

Accepts a floating-point number representing the tangent of an angle and returns the value of said angle in radians

INPUTS

fnum1 - floating-point number

RESULT

fnum2 - floating-point number

BUGS

None

SEE ALSO

## SPCmp

## NAME

SPCmp -- compare two floating-point numbers and set appropriate condition codes

## C USAGE

```
if (SPCmp(fnum1, fnum2)) {...}
 D1 D0
```

## FUNCTION

- Accepts two floating-point numbers and returns the condition codes set to indicate the result of said comparison. Additionally, the integer functional result is returned to indicate the result of said comparison.

## INPUTS

fnum1 - floating-point number  
fnum2 - floating-point number

## RESULT

Condition codes set to reflect the following branches:

```
GT - fnum2 > fnum1
GE - fnum2 >= fnum1
EQ - fnum2 = fnum1
NE - fnum2 != fnum1
LT - fnum2 < fnum1
LE - fnum2 <= fnum1
```

Integer functional result as:

```
+1 => fnum1 > fnum2
-1 => fnum1 < fnum2
0 => fnum1 = fnum2
```

## BUGS

None

## SEE ALSO

## SPCos

## NAME

SPCos -- obtain the cosine of the floating point number

## SYNOPSIS

```
fnum2 = SPCos(fnum1);
 D0
```

## FUNCTION

- Accepts a floating point number representing an angle in radians and returns the cosine of said angle

## INPUTS

fnum1 - floating point number

## RESULT

fnum2 - floating point number

## BUGS

None

## SEE ALSO

SPCosh

NAME

SPCosh -- obtain the hyperbolic cosine of the floating point number

SYNOPSIS

```
fnum2 = SPCosh(fnum1);
 D0
```

FUNCTION

Accepts a floating point number representing an angle in radians and returns the hyperbolic cosine of said angle

INPUTS

fnum1 - floating point number

RESULT

fnum2 - floating point number

BUGS

None

SEE ALSO

SPDiv

NAME

SPDiv -- divide two floating-point numbers

C USAGE

```
fnum3 = SPDiv(fnum1, fnum2);
 D1 D0
```

FUNCTION

Accepts two floating-point numbers and returns the arithmetic division of said numbers.

INPUTS

fnum1 - floating-point number  
fnum2 - floating-point number

RESULT

fnum3 - floating-point number

BUGS

None

SEE ALSO

SPExp

NAME

SPExp -- obtain the exponent (e\*\*X) of the floating-point number

SYNOPSIS

```
fnum2 = SPExp(fnum1);
 D0
```

FUNCTION

Accepts a floating-point number and returns e raised to the input numbers power

INPUTS

fnum1 - floating-point number

RESULT

fnum2 - floating-point number

BUGS

None

SEE ALSO

SPFieee

NAME

SPFieee -- convert an IEEE standard number to FFP format

SYNOPSIS

```
fnum = SPFieee(ieeenum);
 D0
```

FUNCTION

Accepts an IEEE standard format number and returns the same number, only converted into Motorola fast floating-point format

INPUTS

ieeenum - floating-point number (IEEE STD format)

RESULT

fnum - floating-point number (Motorola FFP format)

BUGS

None

SEE ALSO



## SPFlt

### NAME

SPFlt -- convert integer number to fast floating-point

### C USAGE

```
fnum = SPFlt(inum);
DO
```

### FUNCTION

Accepts an integer and returns the converted floating-point result of said number.

### INPUTS

inum - signed integer number

### RESULT

fnum - floating-point number

### BUGS

None

### SEE ALSO

## SplitCList

### NAME

SplitCList -- split a clist

### SYNOPSIS

```
tailCList = SplitCList(cList)
DO A0
```

### FUNCTION

Splits a clist into two clists. The original clist will contain the head of the clist up to but not including the mark (obtained via the MarkCList command). A new clist will be created and returned containing the bytes associated with the mark thru the end of the original clist.

### INPUTS

cList -  
a longword descriptor for a clist that can be used for clist functions.

### RESULTS

tailCList -  
a longword descriptor for a clist that contains the tail end of the original clist.

### EXCEPTIONS

If there is not enough memory to build the new clist or the mark is invalid, tailCList is negative.

SPLog

NAME

SPLog -- obtain the natural logarithm of the floating-point number

SYNOPSIS

```
fnum2 = SPLog(fnum1);
 DO
```

FUNCTION

Accepts a floating-point number and returns the natural logarithm (base e) of said number

INPUTS

fnum1 - floating-point number

RESULT

fnum2 - floating-point number

BUGS

None

SEE ALSO

SPLog10

NAME

SPLog10 -- obtain the naperian logarithm (base 10) of the floating-point number

SYNOPSIS

```
fnum2 = SPLog10(fnum1);
 DO
```

FUNCTION

Accepts a floating-point number and returns the naperian logarithm (base 10) of said number

INPUTS

fnum1 - floating-point number

RESULT

fnum2 - floating-point number

BUGS

None

SEE ALSO

SPMul

NAME

SPMul -- multiply two floating-point numbers

C USAGE

```
fnum3 = SPMul(fnum1, fnum2);
 D1 D0
```

FUNCTION

Accepts two floating-point numbers and returns the arithmetic multiplication of said numbers.

INPUTS

fnum1 - floating-point number  
fnum2 - floating-point number

RESULT

fnum3 - floating-point number

BUGS

None

SEE ALSO

SPNeg

NAME

SPNeg -- negate the supplied floating-point number

C USAGE

```
fnum2 = SPNeg(fnum1);
 D0
```

FUNCTION

Accepts a floating-point number and returns the value of said number after having been subtracted from 0.0

INPUTS

fnum1 - floating-point number

RESULT

fnum2 - floating-point negation of fnum1

BUGS

None

SEE ALSO

SPPow

NAME

SPPow -- obtain the exponentiation of two FFP numbers

SYNOPSIS

```
fnum3 = SPPow(fnum1, fnum2);
 D1 D0
```

FUNCTION

Accepts two (2) floating-point numbers and returns the result of fnum1 raised to the fnum2 power

INPUTS

fnum1 - floating-point number  
fnum2 - floating-point number

RESULT

fnum3 - floating-point number

BUGS

None

SEE ALSO

SPSin

NAME

SPSin -- obtain the sine of the floating-point number

SYNOPSIS

```
fnum2 = SPSin(fnum1);
 D0
```

FUNCTION

Accepts a floating-point number representing an angle in radians and returns the sine of said angle

INPUTS

fnum1 - floating-point number

RESULT

fnum2 - floating-point number

BUGS

None

SEE ALSO

SPSincos

NAME

SPSincos -- obtain the sine & cosine of the FFP number

SYNOPSIS

```
fnum3 = SPSincos(fnum1, &fnum2);
 D1 D0
```

FUNCTION

Accepts a floating-point number representing an angle in radians and returns both the sine & cosine of said angle

INPUTS

fnum1 - floating-point number  
&fnum2 - address of cosine result

RESULT

fnum2 - floating-point number (cosine)  
fnum3 - floating-point number (sine)

BUGS

None

SEE ALSO

SPSinh

NAME

SPSinh -- obtain the hyperbolic sine of the floating-point number

SYNOPSIS

```
fnum2 = SPSinh(fnum1);
 D0
```

FUNCTION

Accepts a floating-point number representing an angle in radians and returns the hyperbolic sine of said angle

INPUTS

fnum1 - floating-point number

RESULT

fnum2 - floating-point number

BUGS

None

SEE ALSO

SPSqrt

NAME

SPSqrt -- obtain the square root of the floating-point number

SYNOPSIS

```
fnum2 = SPSqrt(fnum1);
 D0
```

FUNCTION

Accepts a floating-point number and returns the square root of said number

INPUTS

fnum1 - floating-point number

RESULT

fnum2 - floating-point number

BUGS

None

SEE ALSO

SPSub

NAME

SPSub -- subtract two floating-point numbers

C USAGE

```
fnum3 = SPSub(fnum1, fnum2);
 D1 D0
```

FUNCTION

Accepts two floating-point numbers and returns the arithmetic subtraction of said numbers.

INPUTS

fnum1 - floating-point number  
fnum2 - floating-point number

RESULT

fnum3 - floating-point number

BUGS

None

SEE ALSO

SPTanh

NAME

SPTanh -- obtain the hyperbolic tangent of the floating-point number

SYNOPSIS

```
fnum2 = SPTanh(fnum1);
 D0
```

FUNCTION

Accepts a floating-point number representing an angle in radians and returns the hyperbolic tangent of said angle

INPUTS

fnum1 - floating-point number

RESULT

fnum2 - floating-point number

BUGS

None

SEE ALSO

SPTieee

NAME

SPTieee -- convert an FFP number to IEEE standard format

SYNOPSIS

```
ieeenum = SPTieee(fnum);
```

FUNCTION

Accepts a Motorola fast floating-point number and returns the same number, only converted into IEEE standard format

INPUTS

fnum - floating-point number (Motorola FFP format)

RESULT

ieeenum - floating-point number (IEEE STD format)

BUGS

None

SEE ALSO

## SPTst

### NAME

SPTst -- compare a fast floating-point number against the value zero (0.0) and set the appropriate condition codes

### C USAGE

```
if (!(SPTst(fnum))) {...}
 D1
```

### FUNCTION

Accepts a floating-point number and returns the condition codes set to indicate the result of a comparison against the value of zero (0.0). Additionally, the integer functional result is returned.

### INPUTS

fnum - floating-point number

### RESULT

Condition codes set to reflect the following branches:

```
EQ - fnum = 0.0
NE - fnum != 0.0
PL - fnum >= 0.0
MI - fnum < 0.0
```

Integer functional result as:

```
+1 => fnum > 0.0
-1 => fnum < 0.0
0 => fnum = 0.0
```

### BUGS

None

### SEE ALSO

## SubCList

### NAME

SubCList -- copy a substring from a clist

### SYNOPSIS

```
cList = SubCList(cList, index, length)
D0 A0 D0 D1
```

### FUNCTION

Copies a substring of the cList into a new cList created by this operation. Starts at offset index into the character list and copies for length bytes. The source clist is not altered.

### INPUTS

cList -  
The clist descriptor used to manage this character list, as returned by NewCList or StrToCL.

index -  
The offset in the character list to start copying the substring from. An index of 0 is the first character in the clist.

length -  
The number of bytes to copy.

### RESULTS

cList -  
a longword descriptor for a clist that can be used for clist functions.

### EXCEPTIONS

If cList is negative, not enough space was available for the new clist.

If the substring does not exist for the index and length specified, the resulting clist will be shorter than expected.



## SumLibrary

### NAME

SumLibrary -- compute and check the checksum on a library

### SYNOPSIS

```
SumLibrary(library)
 Al
```

### FUNCTION

SumLibrary computes a new checksum on a library. It can also be used to check an old checksum. If an old checksum does not match and the library has not been marked as changed then the system will alert the user.

### INPUTS

library - a pointer to the library to be changed

### EXCEPTIONS

An alert will occur if the checksum fails.

## SuperState

### NAME

SuperState -- enter supervisor state with user stack

### SYNOPSIS

```
oldSysStack = SuperState()
D0
```

### FUNCTION

Enter supervisor mode while running on the user's stack. The user still has access to user stack variables. Be careful though, the user stack must be large enough to accommodate space for all interrupt data -- this includes all possible nesting of interrupts. This function is a no op when called from supervisor state.

### RESULTS

oldSysStack - system stack pointer  
Save this. It will come in useful when you return to user state. If the system is already in supervisor mode, oldSysStack is zero.

### SEE ALSO

UserState

### SwapBitsRastPortClipRect

#### NAME

SwapBitsRastPortClipRect -- swap bits between common bitmap  
and obscured ClipRect

#### SYNOPSIS

```
SwapBitsRastPortClipRect(rp, cr)
 a0 al
```

#### INPUTS

rp = pointer to rastport  
cr = pointer to cliprect to swap bits with

#### FUNCTION

Support routine useful for those that need to do some operations not done by the layer library. Allows programmer to swap the contents of a small BitMap with a subsection of the display. This is accomplished without using extra memory. The bits in the display RastPort are exchanged with the bits in the ClipRect's BitMap.

#### SEE ALSO

### SyncSBitMap

#### NAME

SyncSBitMap -- synchronize Super BitMap with whatever is  
in the standard Layer bounds

#### SYNOPSIS

```
SyncSBitMap(layer *)
 a0
```

#### FUNCTION

Copies all bits from ClipRects in Layer into Super BitMap BitMap. This is used for those functions that do not want to deal with the ClipRect structures but do want to be able to work with a SuperBitMap Layer.

#### INPUTS

layer is a pointer to a Layer that has a SuperBitMap  
The Layer should already be locked by the caller.

#### SEE ALSO

CopySBitMap

## Text

### NAME

Text -- write text characters (no formatting)

### SYNOPSIS

```
error = Text(RastPort, string, count), gfxLib
D0 A1 A0 D0-0:16 A6
```

### FUNCTION

This graphics function writes printable text characters to the specified RastPort at the current position. No control meaning is applied to any of the characters, and only text on the current line is output.

### INPUTS

RastPort - a pointer to the RastPort which describes where the text is to be output  
count - the string length. If zero, there are no characters to be output.  
string - the address of string to output

### EXCEPTIONS

#### BOUNDS -

If the characters displayed run past the RastPort boundary, the current position is truncated to the boundary, and thus does not represent the true position.

## TextLength

### NAME

TextLength -- determine raster length of text data

### SYNOPSIS

```
length = TextLength(rastPort, string, count)
D0 A1 A0 D0-0:16
```

### FUNCTION

This graphics function determines the length that text data would occupy if output to the specified RastPort with the current attributes. The length is specified as the number of raster dots: to determine what the current position would be after a Write using this string, add the length to cp\_x (cp\_y is unchanged by Write).

### INPUTS

RastPort - a pointer to the RastPort, which describes where the text attributes reside.  
string - the address of string to determine the length of  
count - the string length. If zero, there are no characters in the string.

### RESULTS

length - the number of pixels in x this text would occupy, not including any negative kerning that may take place at the beginning of the text string, nor taking into account the effects of any clipping that may take place.

### BUGS

A length that would overflow single-word arithmetic is not calculated correctly.

## ThinLayerInfo

### NAME

ThinLayerInfo -- convert 1.1 LayerInfo to 1.0 LayerInfo

### SYNOPSIS

```
ThinLayerInfo(li)
 a0
```

### INPUTS

li = pointer to LayerInfo structure

### FUNCTION

Returns the extra memory needed that was allocated with FattenLayerInfo. This must be done prior to freeing the Layer\_Info structure itself. V1.1 software should be using DisposeLayerInfo.

### SEE ALSO

layers.h  
DisposeLayerInfo, FattenLayerInfo

## Translate

### NAME

Translate -- Converts an English string into phonetics

### SYNOPSIS

```
rtnCode = Translate(instring, inlen, outbuf, outlen)
```

### FUNCTION

The translate function converts an English string into a string of phonetic codes suitable as input to the narrator device.

### INPUTS

instring - pointer to English string  
inlen - length of English string  
outbuf - a char array which will hold the phonetic codes  
outlen - the length of the output array

### RESULTS

Translate will return a zero if no error has occurred. The only error that can occur is overflowing the output buffer. If Translate determines that an overflow will occur, it will stop the translation at a word boundary before the overflow happens. If this occurs, Translate will return a negative number whose absolute value indicates where in the INPUT string Translate stopped. The user can then use the offset -rtnCode from the beginning of the buffer in a subsequent Translate call to continue the translation where s/he left off.

### SEE ALSO

## UnGetCLChar

### NAME

UnGetCLChar -- add a byte to the beginning of a character list

### SYNOPSIS

```
error = UnGetCLChar(cList, byte)
D0 A0 D0
```

### FUNCTION

Adds a byte to the beginning of the character list described by the cList.

### INPUTS

cList - The cList header used to manage this character list, as returned by AllocCList or StrToCL.  
byte - The byte to add to the beginning of the character list

### RESULTS

error - non-zero indicates the byte could not be added

## UnGetCLWord

### NAME

UnGetCLWord -- add a word to the beginning of a character list

### SYNOPSIS

```
error = UnGetCLWord(cList, word)
D0 A0 D0
```

### FUNCTION

Adds a word to the beginning of the character list described by the cList.

### INPUTS

cList - The cList header used to manage this character list, as returned by AllocCList or StrToCL.  
word - The word to add to the beginning of the character list

### RESULTS

error - non-zero indicates the number of bytes not added. Partial words are not added, so error is always zero or two.

## UnLoadSeg

### NAME

UnLoadSeg -- unload a segment previously loaded by LoadSeg

### SYNOPSIS

UnLoadSeg( segment )  
DI

### FUNCTION

UnLoadSeg unloads the segment identifier that was returned by LoadSeg.  
'segment' may be zero.

### INPUTS

segment - BCPL pointer to a segment.

## UnLock

### NAME

UnLock -- unlock a directory or file

### SYNOPSIS

UnLock( lock )  
DI

### FUNCTION

UnLock removes a filing system lock obtained from Lock, DupLock, or  
CreateDir.

### INPUTS

lock - BCPL pointer to a lock

## UnlockLayer

### NAME

UnlockLayer -- unlock layer and allow graphics routines to use it.

### SYNOPSIS

```
UnlockLayer(l)
 a0
```

### INPUTS

l = pointer to a layer

### FUNCTION

When finished changing the ClipRects or whatever you were doing with this layer, you must unlock it to allow the other task to proceed with its graphic output.

### SEE ALSO

layers.h

## UnlockLayerInfo

### NAME

UnlockLayerInfo -- unlock the LayerInfo structure.

### SYNOPSIS

```
UnlockLayerInfo(li)
 a0
```

### INPUTS

li = pointer to LayerInfo structure

### FUNCTION

Before doing an operation that requires the LayerInfo structure, makes sure that no other task is also using the LayerInfo structure. This procedure returns when the LayerInfo belongs to this task. There should be an UnlockLayerInfo for every LockLayerInfo.

All layer routines presently LockLayerInfo when they start-up and UnlockLayerInfo as they exit. Programmers will need to use these Lock/Unlock routines if they wish to do something with the layer structure that is not supported by the layer library.

### SEE ALSO

layers.h UnlockLayerInfo()

## UnlockLayerRom

NAME UnlockLayerRom -- unlock Layer structure by rom (gfx.lib) code

SYNOPSIS  
UnlockLayerRom( layer )  
                  a5

### FUNCTION

Decrements lock count and unlocks layer if the result is 0.  
Once the layer is really unlocked the, layerlib may then  
modify this layer.

INPUTS  
layer = pointer to Layer structure

NOTE  
There should be an UnlockLayer for every LockLayer.  
This call does destroy scratch registers.

SEE ALSO  
layers.h, LockLayer()

## UnlockLayers

NAME UnlockLayers -- unlock all layers from graphics output  
Restart graphics output to layers that  
have been waiting

SYNOPSIS  
UnlockLayers( li )  
                  a0

INPUTS  
li = pointer to LayerInfo structure.

FUNCTION  
Make all layers in this layer list unlocked.  
Then call UnlockLayerInfo.

SEE ALSO  
layers.h UnlockLayer()



## UnPutCLChar

NAME  
UnPutCLChar -- get a byte from the end of a character list

SYNOPSIS  
byte = UnPutCLChar(cList)  
D0                    A0

FUNCTION  
Gets a byte from the end of the character list described by the cList.

INPUTS  
cList -  
The cList header used to manage this character list, as returned by AllocCList or StrToCL.

RESULTS  
byte -  
The byte from the end of the character list. If no data is available, the upper three bytes are set (longword is -1).

## UnPutCLWord

NAME  
UnPutCLWord -- get a word from the end of a character list

SYNOPSIS  
word = UnPutCLWord(cList)  
D0                    A0

FUNCTION  
Gets a word from the end of the character list described by the cList.

INPUTS  
cList -  
The cList header used to manage this character list, as returned by AllocCList or StrToCL.

RESULTS  
word -  
The word from the beginning of the character list. If no data is available, the upper two bytes are set (longword is -1). Partial words (1 byte) are not returned.

## UpfrontLayer

### NAME

UpfrontLayer -- put layer in front of all other layers

### SYNOPSIS

```
BOOLEAN UpfrontLayer(li, l)
 a0 al
```

### INPUTS

li = pointer to LayerInfo structure  
l = pointer to a nonbackdrop layer

### FUNCTION

Moves this layer in front of all others, swapping bits in and out of the display with other layers. If this is a refresh layer, collects damage list and sets bit in Flags if redraw required. By clearing the BACKDROP bit in the layers Flags, you may bring a Backdrop layer up to the front of all other layers.

### RETURNS

TRUE if operation successful  
FALSE if operation unsuccessful (probably out of memory)

### SEE ALSO

layers.h

## UserState

### NAME

UserState -- return to user state with user stack

### SYNOPSIS

```
UserState(sysStack)
 D0
```

### FUNCTION

Return to user state with user stack, from supervisor state with user stack. This function is normally used in conjunction with the SuperState function above.

This function must not be called from the user state.

### INPUT

sysStack - supervisor stack pointer

### SEE ALSO

SuperState

## VBeamPos

### NAME

VBeamPos -- get vertical beam position at this instant

### SYNOPSIS

```
pos = VBeamPos()
d0
```

### FUNCTION

Gets the vertical beam position from the hardware.

### INPUTS

None

### RESULT

Interrogates hardware for beam position and returns value.  
valid results in the range of 0-255

### BUGS

Because of hardware constraints, if the vertical beam is  
between 256 and 262, 0 through 6 may be returned.

### NOTE

Because of multitasking, the actual value returned may have  
no use.

## ViewAddress

### NAME

ViewAddress -- return the address of the Intuition View  
structure

### SYNOPSIS

```
ViewAddress()
```

### FUNCTION

This routine returns the address of the Intuition View  
structure. If you want to use any of the graphics, text, or  
animation primitives in your window and that primitive  
requires a pointer to a View, this routine will return the  
address of the View for you.

### INPUTS

None.

### RESULT

Returns the address of the Intuition View structure.

### BUGS

It would be hard for this routine to have a bug.

### SEE ALSO

All of the graphics, text, and animation primitive.

## ViewPortAddress

### NAME

ViewPortAddress -- return the address of a window's ViewPort structure

### SYNOPSIS

```
ViewPortAddress(Window)
 A0
```

### FUNCTION

This routine returns the address of the ViewPort structure associated with the specified window. This is actually the ViewPort of the screen within which the window is displayed. If you want to use any of the graphics, text, or animation primitives in your window and that primitive requires a pointer to a ViewPort structure, you can use this call.

### INPUTS

Window = pointer to the Window structure for which you want the ViewPort address.

### RESULT

Returns the address of the window's ViewPort structure.

### BUGS

It would be hard for this routine to have a bug.

### SEE ALSO

All of the graphics, text, and animation primitives.

## Wait

### NAME

Wait -- wait for one or more signals

### SYNOPSIS

```
signals = Wait(signalSet)
 D0 D0
```

### FUNCTION

This function will cause the current task to suspend waiting for one or more signals. When any of the specified signals occurs, the task will return to the ready state. If a signal occurred prior to calling Wait, the wait condition will be immediately satisfied, and the task will continue to run.

This function cannot be called while in supervisor mode!

### INPUT

signalSet - the set of signals for which to wait.  
Each bit represents a particular signal.

### RESULTS

## WaitBlit

### NAME

WaitBlit -- Waits for the blitter to be finished before proceeding with anything else.

### SYNOPSIS

WaitBlit()

### FUNCTION

WaitBlit returns when the blitter is idle. This function should normally be used only when dealing with the blitter in a synchronous manner, such as when using OwnBlitter and DisownBlitter. WaitBlit does not wait for all blits queued up using QBlit or QBSBlit.

### INPUTS

None

### RESULT

Your program waits until the blitter is finished.

### BUGS

Because of a bug in Agnus, this code may return too soon when the blitter has in fact not started the blit yet, even though BltSize has been written. This most often occurs in a heavily loaded system with extended memory, HIRES, and 4 bitplanes.

### SEE ALSO

OwnBlitter, DisownBlitter

## WaitBOVP

### NAME

WaitBOVP -- wait till vertical beam reaches bottom of this ViewPort.

### SYNOPSIS

WaitBOVP( ViewPort )  
a0

### FUNCTION

Returns when vertical beam reaches bottom of this viewport.

### INPUTS

ViewPort = pointer to ViewPort structure

## WaitForChar

### NAME

WaitForChar -- determine whether characters arrive at a virtual terminal within a time limit

### SYNOPSIS

```
bool = WaitForChar(file, timeout)
D0 D1 D2
```

### FUNCTION

If a character is available to be read from the file associated with the handle 'file' within a certain time, indicated by 'timeout,' WaitForChar returns -1 (TRUE); otherwise, it returns 0 (FALSE). If a character is available, you can use Read to read it. Note that WaitForChar is only valid when the I/O streams are connected to a virtual terminal device. 'timeout' is specified in microseconds.

### INPUTS

file - BCPL pointer to a file handle  
timeout - integer

### RESULTS

bool - boolean

## WaitIO

### NAME

WaitIO -- wait for completion of an I/O request

### SYNOPSIS

```
error = WaitIO(iORequest)
D0 A1
```

### FUNCTION

This function waits for the specified I/O request to complete. If the I/O has already completed, this function will return immediately.

This function should be used with care, as it does not return until the I/O request completes; if the I/O never completes, this function will never return, and your task will hang. If this situation is a possibility, it is safer to use the Wait function, which will return when any particular signal is received. This is how I/O timeouts can be properly handled.

### INPUTS

iORequest - pointer to an I/O request block

### RESULTS

error - zero if successful, else an error is returned

### SEE ALSO

SendIO

## WaitPort

### NAME

WaitPort -- wait for a given port to be non-empty

### SYNOPSIS

```
message = WaitPort(port)
D0 A0
```

### FUNCTION

This function waits for the given port to become non-empty. If necessary, the Wait function will be called to wait for the port signal. If a message is already present at the port, this function will return immediately. The return value is always a pointer to the first message queued (but it is not removed from the queue).

### INPUT

port - a pointer to the message port

### RETURN

message - a pointer to the first available message

### SEE ALSO

GetMsg

## WaitTOF

### NAME

WaitTOF -- wait for the top of the next video frame

### SYNOPSIS

```
WaitTOF()
```

### FUNCTION

Waits for vertical blank to occur and all vertical blank service routines to complete before returning to caller.

### BUGS

### INPUTS

none

## WBenchToBack

### NAME

WBenchToBack -- send the Workbench screen in back of all screens

### SYNOPSIS

WBenchToBack()

### FUNCTION

This routine causes the Workbench screen, if it is currently opened, to go to the background. This does not "move" the screen up or down; it affects only the depth arrangement of the screen.

If the Workbench screen was opened, this function returns TRUE; otherwise, it returns FALSE.

### INPUTS

None

### RESULT

If the Workbench screen was opened, this function returns TRUE; otherwise, it returns FALSE.

### BUGS

Non.

### SEE ALSO

WBenchToFront()

## WBenchToFront

### NAME

WBenchToFront -- bring the Workbench screen in front of all screens

### SYNOPSIS

WBenchToFront()

### FUNCTION

This routine causes the Workbench screen, if it is currently opened, to come to the foreground. This does not "move" the screen up or down; it affects only the depth arrangement of the screen.

If the Workbench screen was opened, this function returns TRUE; otherwise, it returns FALSE.

### INPUTS

None

### RESULT

If the Workbench screen was opened, this function returns TRUE; otherwise, it returns FALSE.

### BUGS

None

### SEE ALSO

WBenchToBack()



## WhichLayer

### NAME

WhichLayer -- in which Layer is this point located?

### SYNOPSIS

```
layer = (struct Layer *)WhichLayer(li, x, y)
 a0 d0 d1
```

### INPUTS

li = pointer to LayerInfo structure  
(x,y) = coordinate in the BitMap

### FUNCTION

Starting at the topmost layer, checks to see if this point (x,y) occurs in this layer. If it does, returns the pointer to this layer. Returns 0 if there is no layer at this point.

### SEE ALSO

layers.h

## WindowLimits

### NAME

WindowLimits -- set the minimum and maximum limits of the window

### SYNOPSIS

```
WindowLimits(Window,MinWidth,MinHeight,MaxWidth,MaxHeight)
 A0 D0 D1 D2 D3
```

### FUNCTION

This routine allows you to adjust the minimum and maximum limits of the window's size. Until this routine is called, the window's size limits are equal to the initial limits specified by the call to OpenWindow().

If you do not want to change any one of the dimensions, set the limit argument for that dimension to zero. If any limit argument is equal to zero, that argument is ignored and the initial setting of that parameter remains undisturbed.

If any argument is out of range (minimums greater than the current size, maximums less than the current size), that limit will be ignored, though the others will still take effect if they are in range. If any argument is out of range, the return value from this procedure will be FALSE. If all arguments are valid, the return value will be TRUE.

If the user is currently sizing this window, the new limits will not take effect until after the sizing is completed.

### INPUTS

Window = pointer to a Window structure.  
MinWidth, MinHeight, MaxWidth, MaxHeight = the new limits for the size of this window. If a limit is set to zero, it will be ignored and that setting will be unchanged.

### RESULT

Returns TRUE if everything was in order. If a parameter was out of range (minimums greater than current size, maximums less than current size), FALSE is returned, and the errant limit request is not fulfilled (though the valid ones will be).

### BUGS

None

### SEE ALSO

OpenWindow()

## WindowToBack

NAME WindowToBack -- ask Intuition to send this window to the back

SYNOPSIS  
WindowToBack(Window)  
A0

FUNCTION  
This routine sends a request to Intuition asking to send the window in back of all other windows in the screen. Note that the window will not be depth arranged immediately; it will be arranged the next time Intuition receives an input event, which happens currently at a minimum rate of ten times per second and a maximum of sixty times a second.

Remember that Backdrop windows cannot be depth-arranged.

INPUTS  
Window = pointer to the structure of the window to be sent to the back.

RESULT  
None

BUGS  
None

SEE ALSO  
MoveWindow(), SizeWindow(), WindowToFront().

## WindowToFront

NAME WindowToFront -- ask Intuition to bring this window to the front

SYNOPSIS  
WindowToFront(Window)  
A0

FUNCTION  
This routine sends a request to Intuition asking to bring the window in front of all other windows in the screen.  
  
Note that the window will not be depth-arranged immediately. It will be arranged the next time Intuition receives an input event, which happens currently at a minimum rate of ten times per second and a maximum of sixty times a second.

Remember that Backdrop windows cannot be depth arranged.

INPUTS  
Window = pointer to the structure of the window to be brought to front.

RESULT  
None

BUGS  
None

SEE ALSO  
MoveWindow(), SizeWindow(), WindowToBack()

## Write

### NAME

Write -- write bytes of data to a file.

### SYNOPSIS

```
returnedLength = Write(file, buffer, length)
D0 D1 D2 D3
```

### FUNCTION

You can copy data with a combination of Read and Write. Write writes bytes of data to the opened file 'file.' 'length' refers to the actual length of data to be transferred; 'buffer' refers to the buffer size.

Write returns a value that indicates the length of information actually written. That is to say, when 'length' is greater than zero, the value of 'length' is the number of characters written. A value of -1 indicates an error. The user of this call must always check for an error return which may, for example, indicate that the disk is full.

### INPUTS

file - BCPL pointer to a file handle  
buffer - address of the first position in the buffer  
length - integer

### RESULTS

returnedLength - integer

## WritePixel

### NAME

WritePixel -- change the pen number of one specific pixel in a specified RasterPort.

### SYNOPSIS

```
WritePixel(rp, x, y)
 a1 D0 D1
```

### FUNCTION

Changes the pen number of the selected pixel in the specified RasterPort to that currently specified by PenA, the primary drawing pen. Obeys DrawModes and minterns in RasterPort.

### INPUTS

x - the X coordinate within the RasterPort at which the selected pixel is located.  
y - the Y coordinate.  
rp - a pointer to the RasterPort to use.

### RESULT

The pixel is changed.

### BUGS

### SEE ALSO

ReadPixel

## XorRectRegion

### NAME

XorRectRegion -- perform second XOR operation of rectangle  
with region, leaving result in region

### SYNOPSIS

```
XorRectRegion(region,rectangle)
 a0 al
```

### Function

Clips away any portion of the region that exists outside  
of the rectangle. Leaves the result in region.

### INPUTS

region = pointer to Region structure  
rectangle = pointer to Rectangle structure

### BUGS

This one does not work yet.



# Appendix B

## Device Summaries

This appendix contains UNIX-like summaries for the commands that may be applied to ROM-resident (or Kickstart-resident) devices, as well as summaries of routines in disk-loadable devices. These documentation files are organized by device. Following this introduction is a listing of each command, followed by the library in which it is located. Note that there are no summaries for the trackdisk device; see the “Trackdisk Device” chapter for information about this device.

The tutorial sections of this manual give you information about how these device commands relate to each other and the prerequisites for calling them. To use any of the device commands, you must first open the device. The correct calling sequence for opening each device is shown in the device tutorial chapter itself. This introduction lists the names of the current set of devices

that are included with the system.

If the device is disk-resident, it is loaded and initialized. The **OpenDevice()** call fills in the **io\_Device** and **io\_Unit** fields of your I/O request block, thereby tying that request block to a specific device. When you say **DoIO(IOResult)**, the **DoIO()** routine, among others, looks in the **IOResult** to find out which device is to be used. This prevents your needing to have a complete (duplicate) set of I/O transmit and control functions for each device.

The following is a list of the names of the devices that are currently a part of the Amiga software. All of these are to be treated as null-terminated strings, which are given to the **OpenDevice()** function. For example:

```
error = OpenDevice("keyboard.device",0,IOResult,0);
```

See **OpenDevice()** in the "Routine Summaries" appendix for the meaning of the various fields of this command.

### Device Names

```
audio.device
clipboard.device
console.device
gameport.device
input.device
keyboard.device
narrator.device
parallel.device
printer.device
serial.device
timer.device
trackdisk.device
```

When you have finished using a device, at the end of your program you should close it, using the **CloseDevice()** function as follows:

```
CloseDevice(IOResult);
```

You must also free whatever memory you may have dedicated to device communication before your program ends. Note that you must make sure that the device has responded to all of your I/O requests by returning your **IOResult** blocks before you attempt to close the device or deallocate the memory.

If the system is running out of memory and needs to free up space, it can check the `accessors` field for various devices. If you have closed the device, it decrements its `accessors` count. For those devices whose `accessors` value is zero, the system can retrieve the memory that the device was using.

Certain devices—the timer and console devices—have routines associated with them. These devices can almost be treated as libraries. To access these routines, you must, as with a library, provide a value to a specific base variable name:

| <b>Device</b> | <b>Base Address Name</b> |
|---------------|--------------------------|
| timer         | TimerBase                |
| console       | ConsoleDevice            |

To get this base address, you must open the device, then copy the `io_Device` field from your **IORequest** block as the base address for this “library” routine. Note that unlike when you are using libraries, you need not issue a **CloseLibrary()** command after using the device routines. The **CloseDevice()** function call is sufficient.

An example showing how to obtain the base address for the timer device is shown in the “Timer Device” chapter in this manual.



## Contents

---

|                 |                  |
|-----------------|------------------|
| AbortIO         | audio.device     |
| AbortIO         | serial.device    |
| AbortIO         | narrator.device  |
| AbortIO         | parallel.device  |
| AddHandler      | input.device     |
| AddResetHandler | keyboard.device  |
| AddTime         | timer.device     |
| ALLOCATE        | audio.device     |
| AskCType        | gameport.device  |
| AskTrigger      | gameport.device  |
| background      | timer.device     |
| BeginIO         | audio.device     |
| BeginIO         | serial.device    |
| BeginIO         | parallel.device  |
| BeginIO         | clipboard.device |
| Break           | serial.device    |
| CDAskKeyMap     | console.device   |
| CDAskKeyMap     | console.device   |
| CDInputHandler  | console.device   |
| CDInputHandler  | console.device   |
| CDSetKeyMap     | console.device   |
| CDSetKeyMap     | console.device   |
| CLEAR           | audio.device     |
| Clear           | input.device     |
| Clear           | serial.device    |
| Clear           | console.device   |
| Clear           | console.device   |
| Clear           | gameport.device  |
| Clear           | keyboard.device  |
| Clear           | parallel.device  |
| Close           | serial.device    |
| Close           | narrator.device  |
| Close           | parallel.device  |
| Close           | clipboard.device |
| CloseDevice     | audio.device     |
| CmpTime         | timer.device     |
| CurrentReadID   | clipboard.device |
| CurrentWriteID  | clipboard.device |
| DumpRPort       | printer.device   |
| Expunge         | audio.device     |
| Expunge         | clipboard.device |
| FINISH          | audio.device     |
| FLUSH           | audio.device     |
| Flush           | serial.device    |
| Flush           | printer.device   |
| Flush           | narrator.device  |
| Flush           | parallel.device  |
| FREE            | audio.device     |
| Invalid         | printer.device   |
| LOCK            | audio.device     |
| Open            | input.device     |
| Open            | serial.device    |
| Open            | gameport.device  |
| Open            | narrator.device  |
| Open            | parallel.device  |
| Open            | clipboard.device |
| OpenDevice      | audio.device     |

|                  |                  |
|------------------|------------------|
| OpenDevice       | console.device   |
| OpenDevice       | console.device   |
| PERVOL           | audio.device     |
| Post             | clipboard.device |
| PrtCommand       | printer.device   |
| Query            | serial.device    |
| Query            | parallel.device  |
| RawKeyConvert    | console.device   |
| RawKeyConvert    | console.device   |
| RawWrite         | printer.device   |
| READ             | audio.device     |
| Read             | serial.device    |
| Read             | console.device   |
| Read             | console.device   |
| Read             | narrator.device  |
| Read             | parallel.device  |
| Read             | clipboard.device |
| ReadEvent        | gameport.device  |
| ReadEvent        | keyboard.device  |
| ReadMatrix       | keyboard.device  |
| RemHandler       | input.device     |
| RemResetHandler  | keyboard.device  |
| RESET            | audio.device     |
| Reset            | input.device     |
| Reset            | serial.device    |
| Reset            | printer.device   |
| Reset            | keyboard.device  |
| Reset            | narrator.device  |
| Reset            | parallel.device  |
| Reset            | clipboard.device |
| ResetHandlerDone | keyboard.device  |
| SetCType         | gameport.device  |
| SetMPort         | input.device     |
| SetMTrig         | input.device     |
| SetMType         | input.device     |
| SetParams        | serial.device    |
| SetParams        | parallel.device  |
| SetPeriod        | input.device     |
| SETPREC          | audio.device     |
| SetThresh        | input.device     |
| SetTrigger       | gameport.device  |
| START            | audio.device     |
| Start            | input.device     |
| Start            | serial.device    |
| Start            | printer.device   |
| Start            | narrator.device  |
| Start            | parallel.device  |
| STOP             | audio.device     |
| Stop             | serial.device    |
| Stop             | printer.device   |
| Stop             | parallel.device  |
| SubTime          | timer.device     |
| TR_ADDREQUEST    | timer.device     |
| TR_GETSYSTIME    | timer.device     |
| TR_SETSYSTIME    | timer.device     |
| UPDATE           | audio.device     |
| Update           | clipboard.device |
| WAITCYCLE        | audio.device     |
| WRITE            | audio.device     |
| Write            | serial.device    |

Write  
Write  
Write  
Write  
Write  
Write  
WriteEvent

console.device  
console.device  
printer.device  
narrator.device  
parallel.device  
clipboard.device  
input.device

Contents

-----  
audio.device/AbortIO  
audio.device/BeginIO  
audio.device/BeginIO/ADCMD\_ALLOCATE  
audio.device/BeginIO/ADCMD\_FINISH  
audio.device/BeginIO/ADCMD\_FREE  
audio.device/BeginIO/ADCMD\_LOCK  
audio.device/BeginIO/ADCMD\_PERVOL  
audio.device/BeginIO/ADCMD\_SETPREC  
audio.device/BeginIO/ADCMD\_WAITCYCLE  
audio.device/BeginIO/CMD\_CLEAR  
audio.device/BeginIO/CMD\_FLUSH  
audio.device/BeginIO/CMD\_READ  
audio.device/BeginIO/CMD\_RESET  
audio.device/BeginIO/CMD\_START  
audio.device/BeginIO/CMD\_STOP  
audio.device/BeginIO/CMD\_UPDATE  
audio.device/BeginIO/CMD\_WRITE  
audio.device/CloseDevice  
audio.device/Expunge  
audio.device/OpenDevice

audio.device/AbortIO

NAME

AbortIO - abort a device command

SYNOPSIS

```
AbortIO(iORequest);
 A1
```

FUNCTION

AbortIO tries to abort a device command. It is allowed to be unsuccessful. If the Abort is successful, the io\_Error field of the iORequest contains an indication that IO was aborted.

INPUTS

iORequest -- pointer to the I/O Request for the command to abort

audio.device/BeginIO

NAME

BeginIO - dispatch a device command

SYNOPSIS

```
BeginIO(iORequest);
 Al
```

FUNCTION

BeginIO has the responsibility of dispatching all device commands. Immediate commands are always called directly, and all other commands are queued to make them single threaded.

INPUTS

iORequest -- pointer to the I/O Request for this command

audio.device/BeginIO/ADCMD\_ALLOCATE

NAME

ADCMD\_ALLOCATE -- allocate a set of audio channels

FUNCTION

ADCMD\_ALLOCATE is a command that allocates multiple audio channels. ADCMD\_ALLOCATE takes an array of possible channel combinations (ioa\_Data) and an allocation precedence (ln\_Pri) and tries to allocate one of the combinations of channels.

If the channel combination array is zero length (ioa\_Length), the allocation succeeds; otherwise, ADCMD\_ALLOCATE checks each combination, one at a time, in the specified order, to find one combination that does not require ADCMD\_ALLOCATE to steal allocated channels.

If it must steal allocated channels, it uses the channel combination that steals the lowest precedence channels. ADCMD\_ALLOCATE cannot steal a channel of equal or greater precedence than the allocation precedence (ln\_Pri).

If it fails to allocate any channel combination and the no-wait flag (ADIOF\_NOWAIT) is set ADCMD\_ALLOCATE returns a zero in the unit field of the I/O request (io\_Unit) and an error (IOERR\_ALLOCFAILED). If the no-wait flag is clear, it places the I/O request in a list that tries to allocate again whenever ADCMD\_FREE frees channels or ADCMD\_SETPREC lowers the channels' precedences.

If the allocation is successful, ADCMD\_ALLOCATE checks if any channels are locked (ADCMD\_LOCK) and if so, replies (ReplyMsg) the lock I/O request with an error (ADIOERR\_CHANNELSTOLEN). Then it places the allocation I/O request in a list waiting for the locked channels to be freed. When all the allocated channels are unlocked, ADCMD\_ALLOCATE:

- . resets (CMD\_RESET) the allocated channels,
- . generates a new allocation key (ioa\_AllocKey), if it is zero,
- . copies the allocation key into each of the allocated channels
- . copies the allocation precedence into each of the allocated channels, and
- . copies the channel bit map into the unit field of the I/O request.

If channels are allocated with a non-zero allocation key, ADCMD\_ALLOCATE allocates with that same key; otherwise, it generates a new and unique key.

ADCMD\_ALLOCATE is synchronous:

- . if the allocation succeeds and there are no locked channels to be stolen, or
- . if the allocation fails and the no-wait flag is set.

In either case, ADCMD\_ALLOCATE replies only (mn\_ReplyPort) if the quick flag (IOF\_QUICK) is clear; otherwise, the allocation is asynchronous, so it clears the quick flag and replies the I/O request after the allocation is finished. If channels are stolen, all audio device commands return an error (IOERR\_NOALLOCATION) when the former user tries to use them again. Do not use ADCMD\_ALLOCATE in interrupt code.

If you decide to store directly to the audio hardware registers, you must either lock the channels you've allocated or set the precedence

to maximum (ADALLOC\_MAXPREC) to prevent the channels from being stolen.

Under all circumstances, unless channels are stolen, you must free (ADCMD\_FREE) all allocated channels when you are finished using them.

#### INPUTS

ln\_Pri - allocation precedence (-128 thru 127)  
mn\_ReplyPort - pointer to message port that receives I/O request after the allocation completes is asynchronous or quick flag (ADIOF\_QUICK) is set  
io\_Device - pointer to device node, must be set by (or copied from I/O block set by) OpenDevice function  
io\_Command - command number for ADCMD\_ALLOCATE  
io\_Flags - flags, must be cleared if not used:  
IOF\_QUICK - (CLEAR) reply I/O request  
(SET) only reply I/O request only if asynchronous (see above text)  
ADIOF\_NOWAIT - (CLEAR) if allocation fails, wait till it succeeds  
(SET) if allocation fails, return error (ADIOERR\_ALLOCFAILED)  
ioa\_AllocKey - allocation key, zero to generate new key; otherwise, it must be set by (or copied from I/O block set by) OpenDevice function or previous ADCMD\_ALLOCATE command  
ioa\_Data - pointer to channel combination options (byte array, bits 0 thru 3 correspond to channels 0 thru 3)  
ioa\_Length - length of the channel combination option array (0 thru 16, 0 always succeeds)

#### OUTPUTS

io\_Unit - bit map of successfully allocated channels (bits 0 thru 3 correspond to channels 0 thru 3)  
io\_Flags - IOF\_QUICK flag cleared if asynchronous (see above text)  
io\_Error - error number:  
0 - no error  
ADIOERR\_ALLOCFAILED - allocation failed

audio.device/BeginIO/ADCMD\_FINISH

#### NAME

ADCMD\_FINISH -- abort writes in progress to audio channels

#### FUNCTION

ADCMD\_FINISH is a command for multiple audio channels. For each selected channel (io\_Unit), if the allocation key (ioa\_AllocKey) is correct and there is a write (CMD\_WRITE) in progress, ADCMD\_FINISH aborts the current write immediately or at the end of the current cycle depending on the sync flag (ADIOF\_SYNC\_CYCLE). If the allocation key is incorrect ADCMD\_FINISH returns an error (ADIOERR\_NOALLOCATION). ADCMD\_FINISH is synchronous and replies only (mn\_ReplyPort) if the quick flag (IOF\_QUICK) is clear. Do not use ADCMD\_FINISH in interrupt code at interrupt level 5 or higher.

#### INPUTS

mn\_ReplyPort - pointer to message port that receives I/O request if the quick flag (IOF\_QUICK) is clear  
io\_Device - pointer to device node, must be set by (or copied from I/O block set by) OpenDevice function  
io\_Unit - bit map of channels to finish (bits 0 thru 3 correspond to channels 0 thru 3)  
io\_Command - command number for ADCMD\_FINISH  
io\_Flags - flags, must be cleared if not used:  
IOF\_QUICK - (CLEAR) reply I/O request  
ADIOF\_SYNC\_CYCLE - (CLEAR) finish immediately  
(SET) finish at the end of current cycle

ioa\_AllocKey - allocation key, must be set by (or copied from I/O block set by) OpenDevice function or ADCMD\_ALLOCATE command

#### OUTPUTS

io\_Unit - bit map of channels successfully finished (bits 0 thru 3 correspond to channels 0 thru 3)  
io\_Error - error number:  
0 - no error  
ADIOERR\_NOALLOCATION - allocation key (ioa\_AllocKey) does not match key for channel

audio.device/BeginIO/ADCMD\_FREE

NAME

ADCMD\_FREE -- free audio channels for allocation

FUNCTION

ADCMD\_FREE is a command for multiple audio channels. For each selected channel (io\_Unit), if the allocation key (ioa\_AllocKey) is correct, ADCMD\_FREE does the following:

- . restores the channel to a known state (CMD\_RESET),
- . changes the channels allocation key, and
- . makes the channel available for reallocation.
- . If the channel is locked (ADCMD\_LOCK) ADCMD\_FREE unlocks it and clears the bit for the channel (io\_Unit) in the lock I/O request. If the lock I/O request has no channel bits set ADCMD\_FREE replies the lock I/O request, and
- . checks if there are allocation requests (ADCMD\_ALLOCATE) waiting for the channel.

Otherwise, ADCMD\_FREE returns an error (ADIOERR\_NOALLOCATION). ADCMD\_FREE is synchronous and replies only (mn\_ReplyPort) if the quick flag (IOF\_QUICK) is clear. Do not use ADCMD\_FREE in interrupt code.

INPUTS

mn\_ReplyPort- pointer to message port that receives I/O request if the quick flag (IOF\_QUICK) is clear

io\_Device - pointer to device node, must be set by (or copied from I/O block set by) OpenDevice function

io\_Unit - bit map of channels to free (bits 0 thru 3 correspond to channels 0 thru 3)

io\_Command - command number for ADCMD\_FREE

io\_Flags - flags, must be cleared if not used:  
IOF\_QUICK - (CLEAR) reply I/O request

ioa\_AllocKey- allocation key, must be set by (or copied from I/O block set by) OpenDevice function or ADCMD\_ALLOCATE command

OUTPUTS

io\_Unit - bit map of channels successfully freed (bits 0 thru 3 correspond to channels 0 thru 3)

io\_Error - error number:  
0 - no error  
ADIOERR\_NOALLOCATION - allocation key (ioa\_AllocKey) does not match key for channel

audio.device/BeginIO/ADCMD\_LOCK

NAME

ADCMD\_LOCK -- prevent audio channels from being stolen

FUNCTION

ADCMD\_LOCK is a command for multiple audio channels. For each selected channel (io\_Unit), if the allocation key (ioa\_AllocKey) is correct, ADCMD\_LOCK locks the channel, preventing subsequent allocations (ADCMD\_ALLOCATE or OpenDevice) from stealing the channel. Otherwise, ADCMD\_LOCK returns an error (ADIOERR\_NOALLOCATION) and will not lock any channels.

Unlike setting the precedence (ADCMD\_SETPREC, ADCMD\_ALLOCATE or OpenDevice) to maximum (ADALLOC\_MAXPREC) which would cause all subsequent allocations to fail, ADCMD\_LOCK causes all higher precedence allocations, even no-wait (ADIOF\_NOWAIT) allocations, to wait until the channels are unlocked.

Locked channels can be unlocked only by freeing them (ADCMD\_FREE), which clears the channel select bits (io\_Unit). ADCMD\_LOCK does not reply the I/O request (mn\_ReplyPort) until all the channels it locks are freed, unless a higher precedence allocation attempts to steal one the locked channels. If a steal occurs, ADCMD\_LOCK replies and returns an error (ADIOERR\_CHANNELSTOLEN). If the lock is replied (mn\_ReplyPort) with this error, the channels should be freed as soon as possible. To avoid a possible deadlock, never make the freeing of stolen channels dependent on another allocations completion.

ADCMD\_LOCK is asynchronous only if the allocation key is correct, in which case it clears the quick flag (IOF\_QUICK); otherwise, it is synchronous and replies only if the quick flag (IOF\_QUICK) is clear. Do not use ADCMD\_LOCK in interrupt code.

INPUTS

mn\_ReplyPort- pointer to message port that receives I/O request if the quick flag (IOF\_QUICK) is clear

io\_Device - pointer to device node, must be set by (or copied from I/O block set by) OpenDevice function

io\_Unit - bit map of channels to lock (bits 0 thru 3 correspond to channels 0 thru 3)

io\_Command - command number for ADCMD\_LOCK

io\_Flags - flags, must be cleared

ioa\_AllocKey- allocation key, must be set by (or copied from I/O block set by) OpenDevice function or ADCMD\_ALLOCATE command

OUTPUTS

io\_Unit - bit map of successfully locked channels (bits 0 thru 3 correspond to channels 0 thru 3) not freed (ADCMD\_FREE)

io\_Flags - IOF\_QUICK flag cleared if the allocation key is correct (no ADIOERR\_NOALLOCATION error)

io\_Error - error number:  
0 - no error  
ADIOERR\_NOALLOCATION - allocation key (ioa\_AllocKey) does not match key for channel  
ADIOERR\_CHANNELSTOLEN- allocation attempting to steal locked channel

audio.device/BeginIO/ADCMD\_PERVOL

NAME

ADCMD\_PERVOL -- change the period and volume for writes in progress to audio channels

FUNCTION

ADCMD\_PERVOL is a command for multiple audio channels. For each selected channel (io\_Unit), if the allocation key (ioa\_AllocKey) is correct and there is a write (CMD\_WRITE) in progress, ADCMD\_PERVOL loads a new volume and period immediately or at the end of the current cycle, depending on the sync flag (ADIOF\_SYNCCYCLE). If the allocation key is incorrect, ADCMD\_PERVOL returns an error (ADIOERR\_NOALLOCATION). ADCMD\_PERVOL is synchronous and replies (mn\_ReplyPort) only if the quick flag (IOF\_QUICK) is clear. Do not use ADCMD\_PERVOL in interrupt code at interrupt level 5 or higher.

INPUTS

mn\_ReplyPort- pointer to message port that receives I/O request if the quick flag (IOF\_QUICK) is clear  
 io\_Device - pointer to device node, must be set by (or copied from I/O block set by) OpenDevice function  
 io\_Unit - bit map of channels to load period and volume (bits 0 thru 3 correspond to channels 0 thru 3)  
 io\_Command - command number for ADCMD\_PERVOL  
 io\_Flags - flags, must be cleared if not used:  
 IOF\_QUICK - (CLEAR) reply I/O request  
 ADIOF\_SYNCCYCLE- (CLEAR) finish immediately (SET) finish at the end of current cycle  
 ioa\_AllocKey- allocation key, must be set by (or copied from I/O block set by) OpenDevice function or ADCMD\_ALLOCATE command  
 ioa\_Period - new sample period in 279.365 ns increments (127 thru 65536, anti-aliasing filter works below 300 to 500 depending on waveform)  
 ioa\_Volume - new volume (0 thru 64, linear)

OUTPUTS

io\_Unit - bit map of channels that successfully loaded period and volume (bits 0 thru 3 correspond to channels 0 thru 3)  
 io\_Error - error number:  
 0 - no error  
 ADIOERR\_NOALLOCATION - allocation key (ioa\_AllocKey) does not match key for channel

audio.device/BeginIO/ADCMD\_SETPREC

NAME

ADCMD\_SETPREC -- set the allocation precedence for audio channels

FUNCTION

ADCMD\_SETPREC is a command for multiple audio channels. For each selected channel (io\_Unit), if the allocation key (ioa\_AllocKey) is correct, ADCMD\_SETPREC sets the allocation precedence to a new value (ln\_Pri) and checks if there are higher-precedence allocation requests (ADCMD\_ALLOCATE) waiting for the channel; otherwise, ADCMD\_SETPREC returns an error (ADIOERR\_NOALLOCATION). ADCMD\_SETPREC is synchronous and replies (mn\_ReplyPort) only if the quick flag (IOF\_QUICK) is clear. Do not use ADCMD\_SETPREC in interrupt code.

INPUTS

ln\_Pri - new allocation precedence (-128 thru 127)  
 mn\_ReplyPort- pointer to message port that receives I/O request if the quick flag (IOF\_QUICK) is clear  
 io\_Device - pointer to device node, must be set by (or copied from I/O block set by) OpenDevice function  
 io\_Unit - bit map of channels to set precedence (bits 0 thru 3 correspond to channels 0 thru 3)  
 io\_Command - command number for ADCMD\_SETPREC  
 io\_Flags - flags, must be cleared if not used:  
 IOF\_QUICK - (CLEAR) reply I/O request  
 ioa\_AllocKey- allocation key, must be set by (or copied from I/O block set by) OpenDevice function or ADCMD\_ALLOCATE command

OUTPUTS

io\_Unit - bit map of channels that successfully set precedence (bits 0 thru 3 correspond to channels 0 thru 3)  
 io\_Error - error number:  
 0 - no error  
 ADIOERR\_NOALLOCATION - allocation key (ioa\_AllocKey) does not match key for channel



audio.device/BeginIO/ADCMD\_WAITCYCLE

NAME

ADCMD\_WAITCYCLE -- wait for an audio channel to complete the current cycle of a write

FUNCTION

ADCMD\_WAITCYCLE is a command for a single audio channel (io\_Unit). If the allocation key (ioa\_AllocKey) is correct and there is a write (CMD\_WRITE) in progress on selected channel, ADCMD\_WAITCYCLE does not reply (mn\_ReplyPort) until the end of the current cycle. If there is no write in progress, ADCMD\_WAITCYCLE replies immediately. If the allocation key is incorrect, ADCMD\_WAITCYCLE returns an error (ADIOERR\_NOALLOCATION). ADCMD\_WAITCYCLE returns an error (IOERR\_ABORTED) if it is canceled (AbortIO) or the channel is stolen (ADCMD\_ALLOCATE). ADCMD\_WAITCYCLE is asynchronous only if it is waiting for a cycle to complete, in which case it clears the quick flag (IOF\_QUICK); otherwise, it is synchronous and replies only if the quick flag (IOF\_QUICK) is clear. Do not use ADCMD\_WAITCYCLE in interrupt code at interrupt level 5 or higher.

INPUTS

mn\_ReplyPort- pointer to message port that receives I/O request, if the quick flag (IOF\_QUICK) is clear, or if a write is in progress on the selected channel and a cycle has completed
io\_Device - pointer to device node, must be set by (or copied from I/O block set by) OpenDevice function
io\_Unit - bit map of channel to wait for cycle (bits 0 thru 3 correspond to channels 0 thru 3). If more than one bit is set, lowest bit number channel is used.
io\_Command - command number for CMD\_WAITCYCLE
io\_Flags - flags, must be cleared if not used:
IOF\_QUICK - (CLEAR) reply I/O request
(SET) reply I/O request only if a write is in progress on the selected channel and a cycle has completed
ioa\_AllocKey- allocation key, must be set by (or copied from I/O block set by) OpenDevice function or ADCMD\_ALLOCATE command

OUTPUTS

io\_Unit - bit map of channel that successfully waited for cycle (bits 0 thru 3 correspond to channels 0 thru 3)
io\_Flags - IOF\_QUICK flag cleared if a write is in progress on the selected channel
io\_Error - error number:
0 - no error
IOERR\_ABORTED - canceled (AbortIO) or channel stolen
ADIOERR\_NOALLOCATION - allocation key (ioa\_AllocKey) does not match key for channel

audio.device/BeginIO/CMD\_CLEAR

NAME

CMD\_CLEAR -- throw away internal caches

FUNCTION

CMD\_CLEAR is a standard command for multiple audio channels. For each selected channel (io\_Unit), if the allocation key (ioa\_AllocKey) is correct, CMD\_CLEAR does nothing; otherwise, CMD\_CLEAR returns an error (ADIOERR\_NOALLOCATION). CMD\_CLEAR is synchronous and replies (mn\_ReplyPort) only if the quick flag (IOF\_QUICK) is clear.

INPUTS

mn\_ReplyPort- pointer to message port that receives I/O request after if the quick flag (IOF\_QUICK) is clear
io\_Device - pointer to device node, must be set by (or copied from I/O block set by) OpenDevice function
io\_Unit - bit map of channels to clear (bits 0 thru 3 correspond to channels 0 thru 3)
io\_Command - command number for CMD\_CLEAR
io\_Flags - flags, must be cleared if not used:
IOF\_QUICK - (CLEAR) reply I/O request
ioa\_AllocKey- allocation key, must be set by (or copied from I/O block set by) OpenDevice function or ADCMD\_ALLOCATE command

OUTPUTS

io\_Unit - bit map of channels successfully cleared (bits 0 thru 3 correspond to channels 0 thru 3)
io\_Error - error number:
0 - no error
ADIOERR\_NOALLOCATION - allocation key (ioa\_AllocKey) does not match key for channel

audio.device/BeginIO/CMD\_FLUSH

NAME

CMD\_FLUSH -- cancel all pending I/O

FUNCTION

CMD\_FLUSH is a standard command for multiple audio channels. For each selected channel (io\_Unit), if the allocation key (ioa\_AllocKey) is correct, CMD\_FLUSH aborts all writes (CMD\_WRITE) in progress or queued and any I/O requests waiting to synchronize with the end of the cycle (ADCMD\_WAITCYCLE); otherwise, CMD\_FLUSH returns an error (ADIOERR\_NOALLOCATION). CMD\_FLUSH is synchronous and replies (mn\_ReplyPort) only if the quick flag (IOF\_QUICK) is clear. Do not use CMD\_FLUSH in interrupt code at interrupt level 5 or higher.

INPUTS

mn\_ReplyPort- pointer to message port that receives I/O request if the quick flag (IOF\_QUICK) is clear  
io\_Device - pointer to device node, must be set by (or copied from I/O block set by) OpenDevice function  
io\_Unit - bit map of channels to flush (bits 0 thru 3 correspond to channels 0 thru 3)  
io\_Command - command number for CMD\_FLUSH  
io\_Flags - flags, must be cleared if not used:  
IOF\_QUICK - (CLEAR) reply I/O request  
ioa\_AllocKey- allocation key, must be set by (or copied from I/O block set by) OpenDevice function or ADCMD\_ALLOCATE command

OUTPUTS

io\_Unit - bit map of channels successfully flushed (bits 0 thru 3 correspond to channels 0 thru 3)  
io\_Error - error number:  
0 - no error  
ADIOERR\_NOALLOCATION - allocation key (ioa\_AllocKey) does not match key for channel

audio.device/BeginIO/CMD\_READ

NAME

CMD\_READ -- normal I/O entry point

FUNCTION

CMD\_READ is a standard command for a single audio channel (io\_Unit). If the allocation key (ioa\_AllocKey) is correct, CMD\_READ returns a pointer (io\_Data) to the I/O block currently writing (CMD\_WRITE) on the selected channel; otherwise, CMD\_READ returns an error (ADIOERR\_NOALLOCATION). If there is no write in progress, CMD\_READ returns zero. CMD\_READ is synchronous and replies (mn\_ReplyPort) only if the quick bit (IOF\_QUICK) is clear.

INPUTS

mn\_ReplyPort- pointer to message port that receives I/O request after if the quick flag (IOF\_QUICK) is clear  
io\_Device - pointer to device node, must be set by (or copied from I/O block set by) OpenDevice function  
io\_Unit - bit map of channel to read (bit 0 thru 3 corresponds to channel 0 thru 3). If more than one bit is set, lowest bit number channel read.  
io\_Command - command number for CMD\_READ  
io\_Flags - flags, must be cleared if not used:  
IOF\_QUICK - (CLEAR) reply I/O request  
ioa\_AllocKey- allocation key, must be set by (or copied from I/O block set by) OpenDevice function or ADCMD\_ALLOCATE command

OUTPUTS

io\_Unit - bit map of channel successfully read (bit 0 thru 3 corresponds to channel 0 thru 3)  
io\_Error - error number:  
0 - no error  
ADIOERR\_NOALLOCATION - allocation key (ioa\_AllocKey) does not match key for channel  
ioa\_Data - pointer to I/O block for current write, zero if none is progress

audio.device/BeginIO/CMD\_RESET

NAME

CMD\_RESET -- restore device to a known state

FUNCTION

CMD\_RESET is a standard command for multiple audio channels. For each selected channel (io\_Unit), if the allocation key (ioa\_AllocKey) is correct, CMD\_RESET:

- . clears the hardware audio registers and attach bits,
- . sets the audio interrupt vector,
- . cancels all pending I/O (CMD\_FLUSH), and
- . unstops the channel if it is stopped (CMD\_STOP),

Otherwise, CMD\_RESET returns an error (ADIOERR\_NOALLOCATION). CMD\_RESET is synchronous and replies (mn\_ReplyPort) only if the quick flag (IOF\_QUICK) is clear. Do not use CMD\_RESET in interrupt code at interrupt level 5 or higher.

INPUTS

- mn\_ReplyPort- pointer to message port that receives I/O request if the quick flag (IOF\_QUICK) is clear
- io\_Device - pointer to device node, must be set by (or copied from I/O block set by) OpenDevice function
- io\_Unit - bit map of channels to reset (bits 0 thru 3 correspond to channels 0 thru 3)
- io\_Command - command number for CMD\_RESET
- io\_Flags - flags, must be cleared if not used:  
IOF\_QUICK - (CLEAR) reply I/O request
- ioa\_AllocKey- allocation key, must be set by (or copied from I/O block set by) OpenDevice function or ADCMD\_ALLOCATE command

OUTPUTS

- io\_Unit - bit map of channels to successfully reset (bits 0 thru 3 correspond to channels 0 thru 3)
- io\_Error - error number:  
0 - no error  
ADIOERR\_NOALLOCATION - allocation key (ioa\_AllocKey) does not match key for channel

audio.device/BeginIO/CMD\_START

NAME

CMD\_START -- start device processing (like ^Q)

FUNCTION

CMD\_START is a standard command for multiple audio channels. For each selected channel (io\_Unit), if the allocation key (ioa\_AllocKey) is correct and the channel was previously stopped (CMD\_STOP), CMD\_START immediately starts all writes (CMD\_WRITE) to the channel. If the allocation key is incorrect, CMD\_START returns an error (ADIOERR\_NOALLOCATION). CMD\_START starts multiple channels simultaneously to minimize distortion if the channels are playing the same waveform and their outputs are mixed. CMD\_START is synchronous and replies (mn\_ReplyPort) only if the quick flag (IOF\_QUICK) is clear. Do not use CMD\_START in interrupt code at interrupt level 5 or higher.

INPUTS

- mn\_ReplyPort- pointer to message port that receives I/O request after if the quick flag (IOF\_QUICK) is clear
- io\_Device - pointer to device node, must be set by (or copied from I/O block set by) OpenDevice function
- io\_Unit - bit map of channels to start (bits 0 thru 3 correspond to channels 0 thru 3)
- io\_Command - command number for CMD\_START
- io\_Flags - flags, must be cleared if not used:  
IOF\_QUICK - (CLEAR) reply I/O request
- ioa\_AllocKey- allocation key, must be set by (or copied from I/O block set by) OpenDevice function or ADCMD\_ALLOCATE command

OUTPUTS

- io\_Unit - bit map of channels successfully started (bits 0 thru 3 correspond to channels 0 thru 3)
- io\_Error - error number:  
0 - no error  
ADIOERR\_NOALLOCATION - allocation key (ioa\_AllocKey) does not match key for channel

## audio.device/BeginIO/CMD\_STOP

## NAME

CMD\_STOP -- stop device processing (like ^S)

## FUNCTION

CMD\_STOP is a standard command for multiple audio channels. For each selected channel (io\_Unit), if the allocation key (ioa\_AllocKey) is correct, CMD\_STOP immediately stops any writes (CMD\_WRITE) in progress; otherwise, CMD\_STOP returns an error (ADIOERR\_NOALLOCATION). CMD\_WRITE queues up writes to a stopped channel until CMD\_START starts the channel or CMD\_RESET resets the channel. CMD\_STOP is synchronous and replies (mn\_ReplyPort) only if the quick flag (IOF\_QUICK) is clear. Do not use CMD\_STOP in interrupt code at interrupt level 5 or higher.

## INPUTS

mn\_ReplyPort- pointer to message port that receives I/O request after if the quick flag (IOF\_QUICK) is clear  
 io\_Device - pointer to device node, must be set by (or copied from I/O block set by) OpenDevice function  
 io\_Unit - bit map of channels to stop (bits 0 thru 3 correspond to channels 0 thru 3)  
 io\_Command - command number for CMD\_STOP  
 io\_Flags - flags, must be cleared if not used:  
 IOF\_QUICK - (CLEAR) reply I/O request  
 ioa\_AllocKey- allocation key, must be set by (or copied from I/O block set by) OpenDevice function or ADCMD\_ALLOCATE command

## OUTPUTS

io\_Unit - bit map of channels successfully stopped (bits 0 thru 3 correspond to channels 0 thru 3)  
 io\_Error - error number:  
 0 - no error  
 ADIOERR\_NOALLOCATION - allocation key (ioa\_AllocKey) does not match key for channel

## audio.device/BeginIO/CMD\_UPDATE

## NAME

CMD\_UPDATE -- force dirty buffers out

## FUNCTION

CMD\_UPDATE is a standard command for multiple audio channels. For each selected channel (io\_Unit), if the allocation key (ioa\_AllocKey) is correct, CMD\_UPDATE does nothing; otherwise, CMD\_UPDATE returns an error (ADIOERR\_NOALLOCATION). CMD\_UPDATE is synchronous and replies (mn\_ReplyPort) only if the quick flag (IOF\_QUICK) is clear.

## INPUTS

mn\_ReplyPort- pointer to message port that receives I/O request after if the quick flag (IOF\_QUICK) is clear  
 io\_Device - pointer to device node, must be set by (or copied from I/O block set by) OpenDevice function  
 io\_Unit - bit map of channels to update (bits 0 thru 3 correspond to channels 0 thru 3)  
 io\_Command - command number for CMD\_UPDATE  
 io\_Flags - flags, must be cleared if not used:  
 IOF\_QUICK - (CLEAR) reply I/O request  
 ioa\_AllocKey- allocation key, must be set by (or copied from I/O block set by) OpenDevice function or ADCMD\_ALLOCATE command

## OUTPUTS

io\_Unit - bit map of channels successfully updated (bits 0 thru 3 correspond to channels 0 thru 3)  
 io\_Error - error number:  
 0 - no error  
 ADIOERR\_NOALLOCATION - allocation key (ioa\_AllocKey) does not match key for channel

audio.device/BeginIO/CMD\_WRITE

NAME

CMD\_WRITE -- normal I/O entry point

FUNCTION

CMD\_WRITE is a standard command for a single audio channel (io\_Unit). If the allocation key (ioa\_AllocKey) is correct, CMD\_WRITE plays a sound using the selected channel; otherwise, it returns an error (ADIOERR\_NOALLOCATION). CMD\_WRITE queues up requests if there is another write in progress or if the channel is stopped (CMD\_STOP). When the write actually starts; if the ADIOF\_PERVOL flag is set, CMD\_WRITE loads volume (ioa\_Volume) and period (ioa\_Period), and if the ADIOF\_WRITEMESSAGE flag is set, CMD\_WRITE replies the write message (ioa\_WriteMsg). CMD\_WRITE returns an error (IOERR\_ABORTED) if it is canceled (AbortIO) or the channel is stolen (ADCMD\_ALLOCATE). CMD\_WRITE is asynchronous only if there is no error, in which case it clears the quick flag (IOF\_QUICK) and replies the I/O request (mn\_ReplyPort) after it finishes writing; otherwise, it is synchronous and replies only if the quick flag (IOF\_QUICK) is clear. Do not use CMD\_WRITE in interrupt code at interrupt level 5 or higher.

INPUTS

mn\_ReplyPort- pointer to message port that receives I/O request after the write completes  
io\_Device - pointer to device node, must be set by (or copied from I/O block set by) OpenDevice function  
io\_Unit - bit map of channel to write (bit 0 thru 3 corresponds to channel 0 thru 3). If more than one bit is set, lowest bit number channel is written.  
io\_Command - command number for CMD\_WRITE  
io\_Flags - flags, must be cleared if not used:  
ADIOF\_PERVOL - (SET) load volume and period  
ADIOF\_WRITEMESSAGE - (SET) reply message at write start  
ioa\_AllocKey- allocation key, must be set by (or copied from I/O block set by) OpenDevice function or ADCMD\_ALLOCATE command  
ioa\_Data - pointer to waveform array (signed bytes (-128 thru 127) in custom chip addressable RAM and word-aligned)  
ioa\_Length - length of the wave array in bytes (2 thru 131072, must be even number)  
ioa\_Period - sample period in 279.365 ns increments (127 thru 65536, anti-aliasing filter works below 300 to 500 depending on waveform), if enabled by ADIOF\_PERVOL  
ioa\_Volume - volume (0 thru 64, linear), if enabled by ADIOF\_PERVOL  
ioa\_Cycles - number of times to repeat array (0 thru 65535, 0 for infinite)  
ioa\_WriteMsg- message replied at start of write, if enabled by ADIOF\_WRITEMESSAGE

OUTPUTS

io\_Unit - bit map of channel successfully written (bit 0 thru 3 corresponds to channel 0 thru 3)  
io\_Flags - IOF\_QUICK flag cleared if there is no error  
io\_Error - error number:  
0 - no error  
IOERR\_ABORTED - canceled (AbortIO) or channel stolen  
ADIOERR\_NOALLOCATION - allocation key (ioa\_AllocKey) does not match key for channel

BUGS

If CMD\_WRITE starts the write immediately after stopping a previous write, you must set the ADIOF\_PERVOL flag or the new data pointer (ioa\_Data) and length (ioa\_Length) may not be loaded.

audio.device/CloseDevice

NAME

CloseDevice - terminate access to the audio device

SYNOPSIS

```
CloseDevice(iORequest);
 Al
```

FUNCTION

The CloseDevice routine notifies the audio device that it will no longer be used. It takes an I/O audio request block (IOAudio) and clears the device pointer (io\_Device). If there are any channels allocated with the same allocation key (ioa\_AllocKey), CloseDevice frees (ADCMD\_FREE) them. CloseDevice decrements the open count, and if it falls to zero and an expunge (Expunge) is pending, the device is expunged.

INPUTS

ioRequest - pointer to audio request block (struct IOAudio)  
io\_Device - pointer to device node, must be set by (or copied from I/O block set by) open (OpenDevice)  
io\_Unit - bit map of channels to free (ADCMD\_FREE) (bits 0 thru 3 correspond to channels 0 thru 3)  
ioa\_AllocKey - allocation key, used to free channels

OUTPUTS

ioRequest - pointer to audio request block (struct IOAudio)  
io\_Device - set to -1  
io\_Unit - set to zero

audio.device/Expunge

NAME

EXPUNGE - indicate a desire to remove the Audio device

FUNCTION

The Expunge routine is called when a user issues a RemDevice call. By the time it is called, the device has already been removed from the device list, so no new opens will succeed. The existence of any other users of the device, as determined by the device open count being non-zero, will cause the Expunge to be deferred. When the device is not in use, or no longer in use, the Expunge is actually performed.

audio.device/OpenDevice

NAME

OpenDevice - open the audio device

SYNOPSIS

error = OpenDevice("audio.device", unitNumber, ioRequest, flags);

FUNCTION

The OpenDevice routine grants access to the audio device. It takes an I/O audio request block (ioRequest), and if it can successfully open the audio device, it loads the device pointer (io\_Device) and the allocation key (ioa\_AllocKey); otherwise, it returns an error (IOERR\_OPENFAIL). OpenDevice increments the open count keeping the device from being expunged (Expunge). If the length (ioa\_Length) is non-zero, OpenDevice tries to allocate (ADCMD\_ALLOCATE) audio channels from a array of channel combination options (ioa\_Data). If the allocation succeeds, the allocated channel combination is loaded into the unit field (ioa\_Unit); otherwise, OpenDevice returns an error (ADIOERR\_ALLOCFALLED). OpenDevice does not wait for allocation to succeed and closes (CloseDevice) the audio device if it fails. To allocate channels, OpenDevice also requires a properly initialized reply port (mn\_ReplyPort) with an allocated signal bit.

INPUTS

unitNumber- not used
ioRequest - pointer to audio request block (struct IOAudio)
ln\_Pri - allocation precedence (-128 thru 127), only necessary for allocation (non-zero length)
mn\_ReplyPort- pointer to message port for allocation, only necessary for allocation (non-zero length)
ioa\_Data - pointer to channel combination options (byte array, bits 0 thru 3 correspond to channels 0 thru 3), only necessary for allocation (non-zero length)
ioa\_Length - length of the channel combination option array (0 thru 16), zero for no allocation
flags - not used

OUTPUTS

ioRequest - pointer to audio request block (struct IOAudio)
io\_Device - pointer to device node if OpenDevice succeeds, otherwise -1
io\_Unit - bit map of successfully allocated channels (bits 0 thru 3 correspond to channels 0 thru 3) if allocation, otherwise 0
io\_Error - error number:
0 - no error
IOERR\_OPENFAIL - open failed
ADIOERR\_ALLOCFALLED - allocation failed, no open
ioa\_AllocKey- unique allocation key, if OpenDevice succeeds
error - copy of io\_Error

Contents

clipboard.device/BeginIO
clipboard.device/CloseDevice
clipboard.device/CLIPREADID
clipboard.device/CLIPWRITEID
clipboard.device/EXPUNGE
clipboard.device/OpenDevice
clipboard.device/POST
clipboard.device/READ
clipboard.device/RESET
clipboard.device/UPDATE
clipboard.device/WRITE

clipboard.device/BeginIO

NAME

BeginIO - initiate clipboard device IO

SYNOPSIS

SendIO(iORequest)  
DoIO(iORequest)

FUNCTION

BeginIO is the workhorse device function used to initiate device commands. It can be called directly or via the Exec library functions SendIO() and DoIO().

clipboard.device/Close

NAME

CloseDevice - terminate access to the clipboard device

SYNOPSIS

CloseDevice(iORequest)

FUNCTION

This routine notifies the clipboard device that the iORequest will no longer be used.



clipboard.device/CLIPREADID

NAME

CLIPREADID - determine the current read identifier

FUNCTION

CLIPREADID fills the io\_ClipID with a clip identifier that can be compared with that of a post command: if greater than the post identifier, the post data held privately by an application is not valid for its own pasting.

IO REQUEST

|            |                                        |
|------------|----------------------------------------|
| io_Message | mn_ReplyPort set up                    |
| io_Device  | preset by OpenDevice                   |
| io_Unit    | preset by OpenDevice                   |
| io_Command | CMD_CLIPREADID                         |
| io_ClipID  | the ClipID of the current write is set |

clipboard.device/CLIPWRITEID

NAME

CLIPWRITEID - determine the current write identifier

FUNCTION

CLIPWRITEID fills the io\_ClipID with a clip identifier that can be compared with that of a post command: if greater than the post identifier, the post is obsolete and need never be satisfied.

IO REQUEST

|            |                                        |
|------------|----------------------------------------|
| io_Message | mn_ReplyPort set up                    |
| io_Device  | preset by OpenDevice                   |
| io_Unit    | preset by OpenDevice                   |
| io_Command | CMD_CLIPWRITEID                        |
| io_ClipID  | the ClipID of the current write is set |

clipboard.device/EXPUNGE

NAME

Expunge - indicate a desire to remove the clipboard device

SYNOPSIS

<Expunge is not generally called by application programs>

FUNCTION

The Expunge routine is called when the system needs the memory used by the clipboard device, and the clipboard device has no open units. The clipboard device is removed from memory until next needed (i.e., until the next OpenDevice("clipboard.device", ...).

clipboard.device/OpenDevice

NAME

OpenDevice - open the clipboard device

SYNOPSIS

OpenDevice("clipboard.device", unit, iORequest, 0)

FUNCTION

The open routine grants access to a device. There are two fields in the iORequest block that will be filled in: io\_Device and io\_Unit.

A successful OpenDevice() call must be matched by a CloseDevice() call when access to the device is no longer needed.

RESULTS

If the open was unsuccessful, returns a non-zero result and the iORequest is not valid.

clipboard.device/POST

NAME

POST - post clip to clipboard

FUNCTION

Indicate to the clipboard device that data is available for use by accessors of the clipboard. This is intended to be used when a cut is large, in a private data format, and/or changing frequently, and it thus makes sense to avoid converting it to an IFF form and writing it to the clipboard unless another application wants it. The post provides a message port to which the clipboard device will send a satisfy message if the data is required.

If the satisfy message is received, the write associated with the post must be performed. The act of writing the clip indicates that the message has been received: it may then be re-used by the clipboard device, and so must actually be removed from the satisfy message port so that the port is not corrupted.

If the application wishes to determine if a post it has performed is still the current clip, it should check the post's io\_ClipID with that returned by the CLIPREADID command. If CLIPREADID is greater, the clip is not still current.

If an application has a pending post and wishes to determine if it should satisfy it (e.g., before it exits), it should check the post's io\_ClipID with that returned by the CLIPWRITEID command. If CLIPWRITEID is greater, there is no need to satisfy the post.

IO REQUEST

|            |                                 |
|------------|---------------------------------|
| io_Message | mn_ReplyPort set up             |
| io_Device  | preset by OpenDevice            |
| io_Unit    | preset by OpenDevice            |
| io_Command | CBD_POST                        |
| io_Data    | pointer to satisfy message port |
| io_ClipID  | zero                            |

RESULTS

|           |                                                                                         |
|-----------|-----------------------------------------------------------------------------------------|
| io_Error  | non-zero if an error occurred                                                           |
| io_ClipID | the clip ID assigned to this post, to be used in the write command if this is satisfied |

clipboard.device/READ

NAME

READ - read clip from clipboard

FUNCTION

The read function serves two purposes.

When io\_Offset is within the clip, it acts as a normal read request, and io\_Data is filled with data from the clipboard. The first read request should have a zero io\_ClipID, which will be filled with the ID assigned for this read. Normal sequential access from the beginning of the clip is achieved by setting io\_Offset to zero for the first read, then leaving it untouched for subsequent reads. If io\_Data is null, then io\_Offset is incremented by io\_Actual as if io\_Length bytes had been read. This is useful for skipping to the end-of-file by using a huge io\_Length.

When io\_Offset is beyond the end of the clip, this acts as a signal to the clipboard device that the application is through reading this clip. Be aware that while an application is in the middle of reading a clip, any attempts to write new data to the clipboard are held off. This read past the end of file indicates that those operations may now be initiated.

IO REQUEST

|            |                                                              |
|------------|--------------------------------------------------------------|
| io_Message | mn_ReplyPort set up                                          |
| io_Device  | preset by OpenDevice                                         |
| io_Unit    | preset by OpenDevice                                         |
| io_Command | CMD_READ                                                     |
| io_Length  | number of bytes to put in data buffer                        |
| io_Data    | pointer to buffer of data to fill, or null to skip over data |
| io_Offset  | byte offset of data to read                                  |
| io_ClipID  | zero if this is the initial read                             |

RESULTS

|           |                                                                                          |
|-----------|------------------------------------------------------------------------------------------|
| io_Error  | non-zero if an error occurred                                                            |
| io_Actual | filled with the actual number of bytes read (the buffer now has io_Actual bytes of data) |
| io_Data   | updated to next read position, which is beyond EOF if io_Actual != io_Length             |
| io_Offset | the clip ID assigned to this read: do not alter for subsequent reads                     |
| io_ClipID |                                                                                          |

clipboard.device/RESET

NAME

RESET - reset the clipboard

FUNCTION

Resets the clipboard device without destroying handles to the open device.

IO REQUEST

|            |                                        |
|------------|----------------------------------------|
| io_Message | mn_ReplyPort set up                    |
| io_Device  | preset by OpenDevice                   |
| io_Command | CMD_RESET                              |
| io_Flags   | IOB_QUICK set if quick I/O is possible |

clipboard.device/UPDATE

NAME

UPDATE - terminate the writing of a cut to the clipboard

FUNCTION

Indicate to the clipboard that the previous write commands are complete and can be used for any pending pastes (reads). This command cannot be issued while any of the write commands are pending.

IO REQUEST

|            |                         |
|------------|-------------------------|
| io_Message | mn_ReplyPort set up     |
| io_Device  | preset by OpenDevice    |
| io_Unit    | preset by OpenDevice    |
| io_Command | CMD_UPDATE              |
| io_ClipID  | the ClipID of the write |

RESULTS

|          |                               |
|----------|-------------------------------|
| io_Error | non-zero if an error occurred |
|----------|-------------------------------|

clipboard.device/WRITE

NAME

WRITE - write clip to clipboard

FUNCTION

This command writes data to the clipboard. This data can be provided sequentially by clearing io\_Offset for the initial write, and using the incremented value unaltered for subsequent writes. If io\_Offset is ever beyond the current clip size, the clip is padded with zeros.

If this write is in response to a SatisfyMsg for a pending post, the io\_ClipID returned by the Post command must be used. Otherwise, a new ID is obtained by clearing the io\_ClipID for the first write. Subsequent writes must not alter the io\_ClipID.

IO REQUEST

|            |                                                                                    |
|------------|------------------------------------------------------------------------------------|
| io_Message | mn_ReplyPort set up                                                                |
| io_Device  | preset by OpenDevice                                                               |
| io_Unit    | preset by OpenDevice                                                               |
| io_Command | CMD_WRITE                                                                          |
| io_Length  | number of bytes from io_Data to write                                              |
| io_Data    | pointer to block of data to write                                                  |
| io_Offset  | usually zero if this is the initial write                                          |
| io_ClipID  | zero if this is the initial write, ClipID of the Post if this is to satisfy a post |

RESULTS

|           |                                                                        |
|-----------|------------------------------------------------------------------------|
| io_Error  | non-zero if an error occurred                                          |
| io_Actual | filled with the actual number of bytes written                         |
| io_Offset | updated to next write position                                         |
| io_ClipID | the clip ID assigned to this write: do not alter for subsequent writes |

Contents

-----  
console.device/CDAskKeyMap  
console.device/CDInputHandler  
console.device/CDSetKeyMap  
console.device/Clear  
console.device/OpenDevice  
console.device/RawKeyConvert  
console.device/Read  
console.device/Write

console.device/CDAskKeyMap

NAME

AskKeyMap - get the current key map structure for this console

FUNCTION

Fills the IO\_DATA buffer with the current KeyMap structure in use by this console unit.

IO REQUEST

|            |                                                                                                    |
|------------|----------------------------------------------------------------------------------------------------|
| io_Message | mn_ReplyPort set if quick I/O is not possible                                                      |
| io_Device  | preset by the call to OpenDevice                                                                   |
| io_Unit    | preset by the call to OpenDevice                                                                   |
| io_Command | CD_ASKKEYMAP                                                                                       |
| io_Flags   | IOF_QUICK if quick I/O possible, else zero                                                         |
| io_Length  | sizeof(*keyMap)                                                                                    |
| io_Data    | struct KeyMap *keyMap<br>eight longwords to describe the raw keycode<br>to byte stream conversion. |

RESULTS

This function sets the error field in the iORequest, and fills the structure at IO\_DATA with the current key map.

console.device/CDInputHandler

NAME

CDInputHandler - handle an input event for the console device

SYNOPSIS

```
CDInputHandler(events, consoleDev)
 A0 A1
```

FUNCTION

Accepts input events from the producer, which is usually the ROM input.task.

NOTES

This function is different from standard device commands in that it is a function in the console device library vectors. The "OpenLibrary" call for the console device is to OpenDevice("console.device", -1, iORequest, 0) and then grab the io\_Device field out of the iORequest as the library vector.

console.device/CDSetKeyMap

NAME

SetKeyMap - set the current key map structure for this console

FUNCTION

Sets the current KeyMap structure used by this console unit to the structure pointed to by IO\_DATA

IO REQUEST

|            |                                                                                                      |
|------------|------------------------------------------------------------------------------------------------------|
| io_Message | mn_ReplyPort set if quick I/O is not possible                                                        |
| io_Device  | preset by the call to OpenDevice                                                                     |
| io_Unit    | preset by the call to OpenDevice                                                                     |
| io_Command | CD_SETKEYMAP                                                                                         |
| io_Flags   | IOF_QUICK if quick I/O possible, else zero                                                           |
| io_Length  | sizeof(*keyMap)                                                                                      |
| io_Data    | struct KeyMap *keyMap<br>eight longwords that describe the raw keycode<br>to byte stream conversion. |

RESULTS

This function sets the error field in the iORequest and fills the current key map from IO\_DATA.

console.device/Clear

NAME

Clear - clear console input buffer

FUNCTION

Remove from the input buffer any reports waiting to satisfy read requests.

IO REQUEST

|            |                                               |
|------------|-----------------------------------------------|
| io_Message | mn_ReplyPort set if quick I/O is not possible |
| io_Device  | preset by the call to OpenDevice              |
| io_Unit    | preset by the call to OpenDevice              |
| io_Command | CMD_CLEAR                                     |
| io_Flags   | IOB_QUICK set if quick I/O is possible        |

## console.device/OpenDevice

### NAME

OpenDevice - a request to open a console device

### SYNOPSIS

```
OpenDevice("console.device", unit, iORequest, 0)
```

### FUNCTION

The open routine grants access to a device. There are two fields in the iORequest block that may be filled in: the IO\_DEVICE field and possibly the IO\_UNIT field.

This open command differs from most other device open commands in that it requires some information to be supplied in the IO\_DATA field of the iORequest block. This initialization information supplies the window that is used by the console device for output.

The unit number that is a standard parameter for an open call is used specially by this device. A unit of -1 indicates that no actual console is to be opened; this is used to get a pointer to the device library vector. A unit of zero binds the supplied window to a unique console. Sharing a console must be done at a level higher than the device. There are no other valid unit numbers.

### IO REQUEST

io\_Data

struct Window \*window

This is the window that will be used for this console. It must be supplied if the unit in the OpenDevice call is 0 (see above). The RPort of this window is potentially in use by the console whenever there is an outstanding write command.

## console.device/RawKeyConvert

### NAME

RawKeyConvert - decode raw input classes

### SYNOPSIS

```
actual = RawKeyConvert(event, buffer, length, keyMap), consoleDev
D0 A0 A1 D1 A2 A6
```

### FUNCTION

This console function converts input events of type IECLASS\_RAWKEY to ANSI bytes, based on the keyMap, and places the result into the buffer.

### INPUTS

event - an InputEvent structure pointer.  
buffer - a byte buffer large enough to hold all anticipated characters generated by this conversion.  
length - maximum anticipation, i.e. the buffer size in bytes.  
keyMap - a KeyMap structure pointer, or null if the default console device key map is to be used.  
consoleDev - the io\_Device of the console device.

### RESULTS

actual - the number of characters in the buffer, or -1 if a buffer overflow was about to occur.

### ERRORS

if actual is -1, a buffer overflow condition was detected. Not all of the characters in the buffer are valid.

### NOTES

This function is different from standard device commands in that it is a function in the console device library vectors. The "OpenLibrary" call for the console device is to OpenDevice("console.device", -1, iORequest, 0), and then grab the io\_Device field out of the iORequest as the library vector.



console.device/Read

NAME

Read - return the next input from the keyboard

FUNCTION

Reads the next input, generally from the keyboard. The form of this input is as an ANSI byte stream: i.e., either ASCII text or control sequences. Raw input events received by the console device can be selectively filtered via the SRE and RRE control sequences (see the write command). Keys are converted via the keymap associated with the unit, which is modified with AskKeyMap and SetKeyMap

If, for example, raw keycodes had been enabled by writing <CSI>ls to the console (where <CSI> is \$9B or Esc[]), keys would return raw keycode reports with the information from the input event itself, in the form:  
<CSI>l0;<keycode>;<modifiers>;0;0;<seconds>;<microseconds>q

If there is no pending input, this command will not be satisfied; if there is some input, but not as much as can fill IO\_LENGTH, the request will be satisfied with the input currently available.

IO REQUEST

io\_Message mn\_ReplyPort set if quick I/O is not possible  
 io\_Device preset by the call to OpenDevice  
 io\_Unit preset by the call to OpenDevice  
 io\_Command CMD\_READ  
 io\_Flags IOF\_QUICK if quick I/O possible, else zero  
 io\_Length sizeof(\*buffer)  
 io\_Data char buffer[]  
 The destination for the characters to read from the keyboard.

RESULTS

This function sets the error field in the iOrequest, and fills the iOrequest IO\_DATA area with the next input, and IO\_ACTUAL with the number of bytes read.

console.device/Write

NAME

Write - write text to the display

FUNCTION

Write a text record to the display. Note that the RPort of the console window is in use while this write command is pending.

IO REQUEST

io\_Message mn\_ReplyPort set if quick I/O is not possible  
 io\_Device preset by the call to OpenDevice  
 io\_Unit preset by the call to OpenDevice  
 io\_Command CMD\_WRITE  
 io\_Flags IOF\_QUICK if quick I/O possible, else zero  
 io\_Length sizeof(\*buffer)  
 io\_Data char buffer[]  
 a buffer containing the ANSI text to write to the console device.

ANSI CODES SUPPORTED

Independent Control Functions (no introducer) --

| Code  | Name | Definition      |
|-------|------|-----------------|
| 00/ 8 | BS   | BACKSPACE       |
| 00/10 | LF   | LINE FEED       |
| 00/11 | VT   | VERTICAL TAB    |
| 00/12 | FF   | FORM FEED       |
| 00/13 | CR   | CARRIAGE RETURN |
| 00/14 | SO   | SHIFT OUT       |
| 00/15 | SI   | SHIFT IN        |
| 01/11 | ESC  | ESCAPE          |

Code or Esc Name Definition

|       |   |     |                                               |
|-------|---|-----|-----------------------------------------------|
| 08/ 4 | D | IND | INDEX: move the active position down one line |
| 08/ 5 | E | NEL | NEXT LINE:                                    |
| 08/13 | M | RI  | REVERSE INDEX:                                |
| 09/11 | [ | CSI | CONTROL SEQUENCE INTRODUCER: see next list    |

ISO-compatible Escape Sequences (introduced by Esc) --

| Esc | Name | Definition                              |
|-----|------|-----------------------------------------|
| a   | INT  | INTERRUPT (will not be supported later) |
| c   | RIS  | RESET TO INITIAL STATE                  |

Control Sequences (introduced by CSI, i.e., \$9B or Esc[]) with parameters: "1" is an optional numeric parameter. "2" is two numeric parameters; e.g., '14;94'. "3" is any number of numeric parameters. Numeric parameters are separated by semicolons.

| Esc[ | Name | Definition       |
|------|------|------------------|
| l@   | ICH  | INSERT CHARACTER |
| lA   | CUU  | CURSOR UP        |
| lB   | CUD  | CURSOR DOWN      |
| lC   | CUF  | CURSOR FORWARD   |
| lD   | CUB  | CURSOR BACKWARD  |

1E CNL CURSOR NEXT LINE  
 1F CPL CURSOR PRECEDING LINE  
 2H CUP CURSOR POSITION  
 1J ED ERASE IN DISPLAY (only to end of display)  
 1K EL ERASE IN LINE (only to end of line)  
 1L IL INSERT LINE  
 1M DL DELETE LINE  
 1P DCH DELETE CHARACTER  
 2R CPR CURSOR POSITION REPORT (in Read stream only)  
 1S SU SCROLL UP  
 1T SD SCROLL DOWN  
 3h SM SET MODE  
 3l RM RESET MODE  
 3m SGR SELECT GRAPHIC RENDITION  
 1n DSR DEVICE STATUS REPORT  
 1t aSLPP SET PAGE LENGTH (private Amiga sequence)  
 1u aSLL SET LINE LENGTH (private Amiga sequence)  
 1x aSLO SET LEFT OFFSET (private Amiga sequence)  
 1y aSTO SET TOP OFFSET (private Amiga sequence)  
 3{ aSRE SET RAW EVENTS (private Amiga sequence)  
 8| aIER INPUT EVENT REPORT (private Amiga Read sequence)  
 3} aRRE RESET RAW EVENTS (private Amiga sequence)  
 1~ aSKR SPECIAL KEY REPORT (private Amiga Read sequence)  
 1 p aSCR SET CURSOR RENDITION (private Amiga sequence)  
 0 q aWSR WINDOW STATUS REQUEST (private Amiga sequence)  
 4 r aWBR WINDOW BOUNDS REPORT (private Amiga Read sequence)

Contents

-----  
 gameport.device/AskCType  
 gameport.device/AskTrigger  
 gameport.device/Clear  
 gameport.device/Open  
 gameport.device/ReadEvent  
 gameport.device/SetCType  
 gameport.device/SetTrigger

gameport.device/AskCTYPE

NAME

AskCTYPE - inquire the current gameport controller type

FUNCTION

This command identifies the type of controller at the game port, so that the signals at the port may be properly interpreted. The controller type has been set by a previous SetCTYPE.

This command always executes immediately.

IO REQUEST

|            |                                                 |
|------------|-------------------------------------------------|
| io_Message | mn_ReplyPort set if quick I/O is not possible   |
| io_Device  | preset by the call to OpenDevice                |
| io_Unit    | preset by the call to OpenDevice                |
| io_Command | GPD_ASKCTYPE                                    |
| io_Flags   | IOB_QUICK set if quick I/O is possible          |
| io_Length  | at least 1                                      |
| io_Data    | the address of the byte variable for the result |

gameport.device/AskTrigger

NAME

AskTrigger - inquire the conditions for a gameport report

FUNCTION

This command inquires what conditions must be met by a game port unit before a pending Read request will be satisfied. These conditions, called triggers, are independent -- that any one occurs is sufficient to queue a gameport report to the Read queue. These conditions are set by SetTrigger.

This command always executes immediately.

IO REQUEST

|            |                                                                       |
|------------|-----------------------------------------------------------------------|
| io_Message | mn_ReplyPort set if quick I/O is not possible                         |
| io_Device  | preset by the call to OpenDevice                                      |
| io_Unit    | preset by the call to OpenDevice                                      |
| io_Command | GPD_ASKTRIGGER                                                        |
| io_Flags   | IOB_QUICK set if quick I/O is possible                                |
| io_Length  | sizeof(gameportTrigger)                                               |
| io_Data    | a structure of type GameportTrigger, which has the following elements |

gpt\_Keys -

GPTB\_DOWNKEYS set if button down transitions trigger a report, and GPTB\_UPKEYS set if button up transitions trigger a report

gpt\_Timeout -

a time which, if exceeded, triggers a report; measured in vertical blank units (60/sec)

gpt\_XDelta -

a distance in x which, if exceeded, triggers a report

gpt\_YDelta -

a distance in y which, if exceeded, triggers a report

gameport.device/Clear

NAME

Clear - clear gameport input buffer

FUNCTION

Removes from the input buffer any gameport reports waiting to satisfy read requests.

IO REQUEST

|            |                                               |
|------------|-----------------------------------------------|
| io_Message | mn_ReplyPort set if quick I/O is not possible |
| io_Device  | preset by the call to OpenDevice              |
| io_Unit    | preset by the call to OpenDevice              |
| io_Command | CMD_CLEAR                                     |
| io_Flags   | IOB_QUICK set if quick I/O is possible        |

gameport.device/Open

NAME

Open - a request to open the GamePort device

SYNOPSIS

OpenDevice("gameport.device", unit, iORequest, 0)

FUNCTION

The open routine grants access to a device. Two fields in the iORequest block will be filled in: the IO\_DEVICE field and the IO\_UNIT field.

The device open count will be incremented. The device cannot be expunged unless this open is matched by a Close device.

INPUTS

|      |                                                |
|------|------------------------------------------------|
| unit | -                                              |
| 0    | unit associated with left gameport controller  |
| 1    | unit associated with right gameport controller |

RESULTS

If the open was unsuccessful, IO\_ERROR will be set, IO\_UNIT and IO\_DEVICE will not be valid.

## gameport.device/ReadEvent

### NAME

ReadEvent - return the next gameport event.

### FUNCTION

Reads gameport events from the gameport and puts them in the data area of the iORequest. If there are no pending gameport events, this command will not be satisfied, but if there are some events, but not as many as can fill IO\_LENGTH, the request will be satisfied with those currently available.

### IO REQUEST

|              |                                                                                                                       |
|--------------|-----------------------------------------------------------------------------------------------------------------------|
| io_Message   | mn_ReplyPort set if quick I/O is not possible                                                                         |
| io_Device    | preset by the call to OpenDevice                                                                                      |
| io_Unit      | preset by the call to OpenDevice                                                                                      |
| io_Command   | GPD_READEVENT                                                                                                         |
| io_Flags     | IOB_QUICK set if quick I/O is possible                                                                                |
| io_Length    | the size of the io_Data area in bytes: there are sizeof(inputEvent) bytes per input event.                            |
| io_Data      | a buffer area to fill with input events. The fields of the input event are:                                           |
| ie_NextEvent | links the events returned                                                                                             |
| ie_Class     | is IECLASS_RAWMOUSE                                                                                                   |
| ie_SubClass  | is 0 for the left, 1 for the right gameport                                                                           |
| ie_Code      | contains any gameport button reports. No report is indicated by the value 0xff.                                       |
| ie_Qualifier | only the relative and button bits are set                                                                             |
| ie_X, ie_Y   | the x and y values for this report, in either relative or absolute device dependent units.                            |
| ie_TimeStamp | the delta time since the last report, given not as a standard timestamp, but as the frame count in the TV_SECS field. |

### RESULTS

This function sets the error field in the iORequest and fills the iORequest with the next gameport events (but not partial events).

### SEE ALSO

gameport.device/SetCType, gameport.device/SetTrigger

## gameport.device/SetCType

### NAME

SetCType - set the current gameport controller type

### FUNCTION

This command sets the type of device at the gameport, so that the signals at the port may be properly interpreted. The port can also be turned off, so that no reports are generated.

This command always executes immediately.

### IO REQUEST

|            |                                                                                                                  |
|------------|------------------------------------------------------------------------------------------------------------------|
| io_Message | mn_ReplyPort set if quick I/O is not possible                                                                    |
| io_Device  | preset by the call to OpenDevice                                                                                 |
| io_Unit    | preset by the call to OpenDevice                                                                                 |
| io_Command | GPD_SETCTYPE                                                                                                     |
| io_Flags   | IOB_QUICK set if quick I/O is possible                                                                           |
| io_Length  | 1                                                                                                                |
| io_Data    | the address of the byte variable describing the controller type, as per the equates in the gameport include file |

gameport.device/SetTrigger

Contents

NAME

SetTrigger - set the conditions for a gameport report

FUNCTION

This command sets what conditions must be met by a game port unit before a pending Read request will be satisfied. These conditions, called triggers, are independent -- that any one occurs is sufficient to queue a gameport report to the Read queue. These conditions are inquired with AskTrigger.

This command always executes immediately.

IO REQUEST

io\_Message     mn\_ReplyPort set if quick I/O is not possible  
io\_Device     preset by the call to OpenDevice  
io\_Unit        preset by the call to OpenDevice  
io\_Command     GPD\_SETTRIGGER  
io\_Flags       IOB\_QUICK set if quick I/O is possible  
io\_Length      sizeof(gameportTrigger)  
io\_Data        a structure of type GameportTrigger, which has the following elements

gpt\_Keys -

GPTB\_DOWNKEYS set if button down transitions trigger a report, and GPTB\_UPKEYS set if button up transitions trigger a report

gpt\_Timeout -

a time which, if exceeded, triggers a report; measured in vertical blank units (60/sec)

gpt\_XDelta -

a distance in x which, if exceeded, triggers a report

gpt\_YDelta -

a distance in y which, if exceeded, triggers a report

input.device/AddHandler  
input.device/Clear  
input.device/Open  
input.device/RemHandler  
input.device/Reset  
input.device/SetMPort  
input.device/SetMTrig  
input.device/SetMType  
input.device/SetPeriod  
input.device/SetThresh  
input.device/Start  
input.device/WriteEvent

input.device/AddHandler

NAME

AddHandler - add an input handler to the device

FUNCTION

Adds a function to the list of functions called to handle input events generated by this device. The function is called as

```
newInputEvents = Handler(inputEvents, handlerData);
D0 A0 A1
```

IO REQUEST

|            |                                         |
|------------|-----------------------------------------|
| io_Message | mn_ReplyPort set                        |
| io_Device  | preset by OpenDevice                    |
| io_Unit    | preset by OpenDevice                    |
| io_Command | IND_ADDHANDLER                          |
| io_Data    | a pointer to an interrupt structure.    |
| is_Data    | the handlerData pointer described above |
| is_Code    | the Handler function address            |

NOTES

The interrupt structure is kept by the input device until a RemHandler command is satisfied for it.

input.device/Clear

NAME

Clear - clear input buffer

FUNCTION

Removes from input buffers any input reports waiting to satisfy read requests.

IO REQUEST

|            |                                               |
|------------|-----------------------------------------------|
| io_Message | mn_ReplyPort set if quick I/O is not possible |
| io_Device  | preset by the call to OpenDevice              |
| io_Unit    | preset by the call to OpenDevice              |
| io_Command | CMD_CLEAR                                     |
| io_Flags   | IOB_QUICK set if quick I/O is possible        |

## input.device/Open

### NAME

Open - a request to open the input device

### SYNOPSIS

OpenDevice("input.device", 0, iORequest, 0)

### FUNCTION

The open routine grants access to a device. Two fields in the iORequest block will be filled in: the IO\_DEVICE field and the IO\_UNIT field.

The device open count will be incremented. The device cannot be expunged unless this open is matched by a CloseDevice.

### RESULTS

If the open was unsuccessful, IO\_ERROR will be set, IO\_UNIT and IO\_DEVICE will not be valid.

## input.device/RemHandler

### NAME

RemHandler - remove an input handler from the device

### FUNCTION

Removes a function previously added to the list of handler functions.

### IO REQUEST

|            |                                       |
|------------|---------------------------------------|
| io_Message | mn_ReplyPort set                      |
| io_Device  | preset by OpenDevice                  |
| io_Unit    | preset by OpenDevice                  |
| io_Command | IND_REMHANDLER                        |
| io_Data    | a pointer to the interrupt structure. |

### NOTES

This command is not immediate



input.device/Reset

NAME

Reset - reset the input

FUNCTION

Reset resets the keyboard device without destroying handles to the open device.

IO REQUEST

|            |                                               |
|------------|-----------------------------------------------|
| io_Message | mn_ReplyPort set if quick I/O is not possible |
| io_Device  | preset by the call to OpenDevice              |
| io_Unit    | preset by the call to OpenDevice              |
| io_Command | CMD_RESET                                     |
| io_Flags   | IOB_QUICK set if quick I/O is possible        |

input.device/SetMPort

NAME

SetMPort - set the current mouse port

FUNCTION

This command sets the gameport port at which the mouse is connected.

IO REQUEST

|            |                                                                                                                                                        |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| io_Message | mn_ReplyPort set if quick I/O is not possible                                                                                                          |
| io_Device  | preset by the call to OpenDevice                                                                                                                       |
| io_Unit    | preset by the call to OpenDevice                                                                                                                       |
| io_Command | IND_SETMPORT                                                                                                                                           |
| io_Flags   | IOB_QUICK set if quick I/O is possible                                                                                                                 |
| io_Length  | 1                                                                                                                                                      |
| io_Data    | a pointer to a byte that is either 0 or 1, indicating that mouse input should be obtained from either the left or right controller port, respectively. |

input.device/SetMTrig

NAME

SetMTrig - set the conditions for a mouse port report

FUNCTION

This command sets what conditions must be met by a mouse before a pending Read request will be satisfied. The trigger specification is that used by the gameport device.

IO REQUEST

|            |                                                                       |
|------------|-----------------------------------------------------------------------|
| io_Message | mn_ReplyPort set if quick I/O is not possible                         |
| io_Device  | preset by the call to OpenDevice                                      |
| io_Unit    | preset by the call to OpenDevice                                      |
| io_Command | IND_SETTRIGGER                                                        |
| io_Flags   | IOB_QUICK set if quick I/O is possible                                |
| io_Length  | sizeof(gameportTrigger)                                               |
| io_Data    | a structure of type GameportTrigger, which has the following elements |

gpt\_Keys -  
GPTB\_DOWNKEYS set if button-down transitions trigger a report, and GPTB\_UPKEYS set if button up transitions trigger a report

gpt\_Timeout -  
a time which, if exceeded, triggers a report; measured in vertical blank units (60/sec)

gpt\_XDelta -  
a distance in x which, if exceeded, triggers a report

gpt\_YDelta -  
a distance in y which, if exceeded, triggers a report

input.device/SetMType

NAME

SetMType - set the current mouse port controller type

FUNCTION

This command sets the type of device at the mouse port, so the signals at the port may be properly interpreted.

IO REQUEST

|            |                                                                                                                  |
|------------|------------------------------------------------------------------------------------------------------------------|
| io_Message | mn_ReplyPort set if quick I/O is not possible                                                                    |
| io_Device  | preset by the call to OpenDevice                                                                                 |
| io_Unit    | preset by the call to OpenDevice                                                                                 |
| io_Command | IND_SETMTYPE                                                                                                     |
| io_Flags   | IOB_QUICK set if quick I/O is possible                                                                           |
| io_Length  | 1                                                                                                                |
| io_Data    | the address of the byte variable describing the controller type, as per the equates in the gameport include file |

input.device/SetPeriod

NAME

SetPeriod - set the key repeat period

FUNCTION

This command sets the period at which a repeating key repeats.  
This command always executes immediately.

IO REQUEST - a timerequest

|             |                                               |
|-------------|-----------------------------------------------|
| io_Message  | mn_ReplyPort set if quick I/O is not possible |
| io_Device   | preset by the call to OpenDevice              |
| io_Unit     | preset by the call to OpenDevice              |
| io_Command  | IND_SETPERIOD                                 |
| io_Flags    | IOB_QUICK set if quick I/O is possible        |
| io_tv_Secs  | the repeat period seconds                     |
| io_tv_Micro | the repeat period microseconds                |

input.device/SetThresh

NAME

SetThresh - set the key repeat threshold

FUNCTION

This command sets the time that a key must be held down before it can repeat. The repeatability of a key may be restricted (as, for example, are the shift keys).

This command always executes immediately.

IO REQUEST - a timerequest

|             |                                               |
|-------------|-----------------------------------------------|
| io_Message  | mn_ReplyPort set if quick I/O is not possible |
| io_Device   | preset by the call to OpenDevice              |
| io_Unit     | preset by the call to OpenDevice              |
| io_Command  | IND_SETHRESH                                  |
| io_Flags    | IOB_QUICK set if quick I/O is possible        |
| io_tv_Secs  | the threshold seconds                         |
| io_tv_Micro | the threshold microseconds                    |

input.device/Start

NAME

Start - restart after stop

FUNCTION

Start restarts the unit after a stop command.

IO REQUEST

|            |                                               |
|------------|-----------------------------------------------|
| io_Message | mn_ReplyPort set if quick I/O is not possible |
| io_Device  | preset by the call to OpenDevice              |
| io_Unit    | preset by the call to OpenDevice              |
| io_Command | CMD_START                                     |
| io_Flags   | IOB_QUICK set if quick I/O is possible        |

input.device/WriteEvent

NAME

WriteEvent - propagate input event(s) to all handlers

FUNCTION

IO REQUEST

|              |                                                                                            |
|--------------|--------------------------------------------------------------------------------------------|
| io_Message   | mn_ReplyPort set if quick I/O is not possible                                              |
| io_Device    | preset by the call to OpenDevice                                                           |
| io_Unit      | preset by the call to OpenDevice                                                           |
| io_Command   | IND_WRITEEVENT                                                                             |
| io_Flags     | IOB_QUICK set if quick I/O is possible                                                     |
| io_Length    | the size of the io_Data area in bytes: there are sizeof(inputEvent) bytes per input event. |
| io_Data      | a buffer area with input event(s). The fields of the input event are:                      |
| ie_NextEvent | links the events together, the last event has a zero ie_NextEvent.                         |
| ie_Class     |                                                                                            |
| ie_SubClass  |                                                                                            |
| ie_Code      |                                                                                            |
| ie_Qualifier |                                                                                            |
| ie_X, ie_Y   |                                                                                            |
| ie_TimeStamp | as desired                                                                                 |

NOTES

The contents of the input event(s) are destroyed.

Contents

keyboard.device/AddResetHandler  
keyboard.device/Clear  
keyboard.device/ReadEvent  
keyboard.device/ReadMatrix  
keyboard.device/RemResetHandler  
keyboard.device/Reset  
keyboard.device/ResetHandlerDone

keyboard.device/AddResetHandler

NAME  
AddResetHandler - add a reset handler to the device

FUNCTION  
Adds a function to the list of functions called to clean up before a hard reset:

```
Handler(handlerData);
Al
```

IO REQUEST

|            |                                         |
|------------|-----------------------------------------|
| io_Message | mn_ReplyPort set                        |
| io_Device  | preset by OpenDevice                    |
| io_Unit    | preset by OpenDevice                    |
| io_Command | KBD_ADDRESETHANDLER                     |
| io_Data    | a pointer to an interrupt structure.    |
| is_Data    | the handlerData pointer described above |
| is_Code    | the Handler function address            |

NOTES  
The interrupt structure is kept by the keyboard device until a RemResetHandler command is satisfied for it.

keyboard.device/Clear

NAME

Clear - clear keyboard input buffer

FUNCTION

Removes from the input buffer any keys transitions waiting to satisfy read requests.

IO REQUEST

|            |                                               |
|------------|-----------------------------------------------|
| io_Message | mn_ReplyPort set if quick I/O is not possible |
| io_Device  | preset by the call to OpenDevice              |
| io_Command | CMD_CLEAR                                     |
| io_Flags   | IOB_QUICK set if quick I/O is possible        |

keyboard.device/ReadEvent

NAME

ReadEvent - return the next keyboard event.

FUNCTION

Read raw keyboard events from the keyboard and put them in the data area of the IORequest. If there are no pending keyboard events, this command will not be satisfied. If there are some events, but not as many as can fill IO\_LENGTH, the request will be satisfied with those currently available.

IO REQUEST

|                                       |                                                                                                                                                                        |
|---------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| io_Message                            | mn_ReplyPort set if quick I/O is not possible                                                                                                                          |
| io_Device                             | preset by the call to OpenDevice                                                                                                                                       |
| io_Command                            | KBD_READEVENT                                                                                                                                                          |
| io_Flags                              | IOB_QUICK set if quick I/O is possible                                                                                                                                 |
| io_Length                             | the size of the io_Data area in bytes; there are sizeof(inputEvent) bytes per input event. a buffer area to fill with input events. The fields of the input event are: |
| io_Data                               |                                                                                                                                                                        |
| ie_NextEvent                          | links the events returned                                                                                                                                              |
| ie_Class                              | is IECLASS_RAWKEY                                                                                                                                                      |
| ie_Code                               | contains the next key up/down reports                                                                                                                                  |
| ie_Qualifier                          | only the shift and numeric pad bits are set                                                                                                                            |
| ie_SubClass, ie_X, ie_Y, ie_TimeStamp | are not used, and set to zero                                                                                                                                          |

RESULTS

This function sets the error field in the IORequest and fills the IORequest with the next keyboard events (but not partial events).

keyboard.device/ReadMatrix

NAME

ReadMatrix - read the current keyboard key matrix

FUNCTION

This function reads the up/down state of every key in the key matrix.

IO REQUEST

|            |                                                                                                                                                                                                     |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| io_Message | mn_ReplyPort set if quick I/O is not possible                                                                                                                                                       |
| io_Device  | preset by the call to OpenDevice                                                                                                                                                                    |
| io_Command | KBD_READMATRIX                                                                                                                                                                                      |
| io_Flags   | IOB_QUICK set if quick I/O is possible                                                                                                                                                              |
| io_Length  | the size of the io_Data area in bytes: this must be big enough to hold the key matrix.                                                                                                              |
| io_Data    | a buffer area to fill with the key matrix: an array of bytes whose component bits reflect each keys state: the state of the key for keycode n is at bit (n MOD 8) in byte (n DIV 8) of this matrix. |

RESULTS

This function sets the error field in the IORequest and sets matrix to the current key matrix.

keyboard.device/RemResetHandler

NAME

RemResetHandler - remove a reset handler from the device

FUNCTION

Removes a function previously added to the list of handler functions.

IO REQUEST

|            |                                               |
|------------|-----------------------------------------------|
| io_Message | mn_ReplyPort set                              |
| io_Device  | preset by OpenDevice                          |
| io_Unit    | preset by OpenDevice                          |
| io_Command | KBD_REMRESETHANDLER                           |
| io_Data    | a pointer to the handler interrupt structure. |

keyboard.device/Reset

NAME

Reset - reset the keyboard

FUNCTION

Reset resets the keyboard device without destroying handles to the open device.

IO REQUEST

|            |                                               |
|------------|-----------------------------------------------|
| io_Message | mn_ReplyPort set if quick I/O is not possible |
| io_Device  | preset by the call to OpenDevice              |
| io_Command | CMD_RESET                                     |
| io_Flags   | IOB_QUICK set if quick I/O is possible        |

keyboard.device/ResetHandlerDone

NAME

ResetHandlerDone - indicate that reset can occur

FUNCTION

Indicates that reset clean-up associated with the handler has completed.

IO REQUEST

|            |                                               |
|------------|-----------------------------------------------|
| io_Message | mn_ReplyPort set                              |
| io_Device  | preset by OpenDevice                          |
| io_Unit    | preset by OpenDevice                          |
| io_Command | KBD_RESETHANDLERDONE                          |
| io_Data    | a pointer to the handler interrupt structure. |



Contents

narrator.device/AbortIO  
narrator.device/CloseDevice  
narrator.device/Flush  
narrator.device/OpenDevice  
narrator.device/Read  
narrator.device/Reset  
narrator.device/Start/Stop  
narrator.device/Write

narrator.device/AbortIO

NAME

AbortIO - abort an IO request

SYNOPSIS

AbortIO(IORequest)

FUNCTION

Aborts a speech IO request. The request may be in the queue or currently active.

INPUTS

IORB of request to abort.

RESULTS

io\_Error field of IORB set to IOERR\_ABORTED

SEE ALSO

narrator.device/CloseDevice

NAME

CloseDevice - terminate access to the narrator device

SYNOPSIS

CloseDevice(IOResult)

FUNCTION

Close invalidates the IO\_UNIT and IO\_DEVICE fields in the IORB, preventing subsequent IO until another OpenDevice. CloseDevice also reduces the open count. If the count goes to 0 and the expunge bit is set, the device is expunged. If the open count goes to zero and the delayed expunge bit is not set, CloseDevice sets the expunge bit.

INPUTS

IOResult block

RESULTS

IOResult block with unit and device pointers invalidated.

SEE ALSO

narrator.device/Flush

NAME

Flush - abort all in-progress and queued requests

SYNOPSIS

Standard device command. See DoIO()/SendIO().

FUNCTION

Aborts all in-progress and queued speech requests.

INPUTS

io\_Command - CMD\_FLUSH

RESULTS

SEE ALSO

## narrator.device/Open

### NAME

OpenDevice - open the narrator device

### SYNOPSIS

```
error = OpenDevice("narrator.device", 0, IORequest, 0);
```

### FUNCTION

The OpenDevice routine grants access to the narrator device. OpenDevice checks the unit number, and if non-zero, returns an error (ND\_UnitErr). If this is the first time the driver has been opened, OpenDevice will attempt to open the audio device and allocate the driver's static buffers. If either of these operations fail, an error is returned (see the .h and .i files for possible error return codes). Next, OpenDevice (done for all opens, not just the first one) initializes the user's IORequest block (IORB). Default values for sex, rate, pitch, pitch mode, sampling frequency, and mouths are set in the appropriate fields of the IORB. Note that if users wish to use non-default values for these parms, the values must be set after the open is done. OpenDevice then assigns a pseudo-unit number to the IORB for use in synchronizing read and write requests. See the read command for more details. Finally, OpenDevice stores the device node pointer in the IORB and clears the delayed expunge bit.

### INPUTS

deviceName - must be "narrator.device"  
unitNumber - must be 0  
IORequest - the user's IORB (need not be initialized)  
flags - not used

### RESULTS

IORB fields set:  
rate - 150 words/minute  
pitch - 110 Hz  
mode - Natural  
sex - Male  
mouths - Off  
sampfreq - 22200  
volume - 64 (max)

error - same as io\_Error field of IORB

SEE ALSO

## narrator.device/Read

### NAME

Read - return the next different mouth shape from an associated write

### SYNOPSIS

Standard device command. See DoIO/SendIO.

### FUNCTION

The read command of the narrator device returns mouth shapes to the user. The shape returned is guaranteed to be different from the previously returned shape (allowing updating to be done only when something has changed). Each read request is associated with a write request by the pseudo-unit number assigned by the OpenDevice call. Since the first structure in the read-mouth IORB is a narrator (write) IORB, this association is easily made by copying the narrator IORB into the narrate\_rb field of the read IORB. See the .hi files. If there is no write in progress or in the device input queue with the same pseudo-unit number as the read request, the read will be returned to the user with an error. This is also how the user knows that the write request has finished and that s/he should not issue any more reads. Note that in this case the mouth shapes may not be different from previously returned values.

### INPUTS

IORB with the narrator\_rb structure copied from the associated write request except for:  
io\_Message - message port for read request  
io\_Command - CMD\_READ  
io\_Error - 0  
width - 0  
height - 0

### RESULTS

IORB fields set:  
width - mouth width in millimeters/3.67  
(division done for scaling)  
height - mouth height in millimeters  
shape - compressed form of mouth shapes  
(internal use only)

SEE ALSO

Write command.

narrator.device/Reset

NAME

Reset - reset the device to a known state

SYNOPSIS

Standard device command. See DoIO()/SendIO().

FUNCTION

Resets the device as though it has just be initialized.  
Aborts all read/write requests whether they are active or enqueued.  
Restarts device if it has been stopped.

INPUTS

io\_Command = CMD\_RESET

RESULTS

SEE ALSO

narrator.device/Start/Stop

NAME

Stop - stops the device  
Start - restarts the device after Stop

SYNOPSIS

Standard device commands. See DoIO()/SendIO().

FUNCTION

Stop halts the currently active speech (if any) and prevents any queued requests from starting.

Start restarts the currently active speech (if any) and allows queued request to start.

INPUTS

io\_Command = CMD\_STOP or CMD\_START

RESULTS

SEE ALSO

narrator.device/Write

NAME

Write - send speech request to the narrator device

SYNOPSIS

Standard device command. See DoIO()/SendIO().

FUNCTION

Performs the speech request. If there is an associated read request on the device input queue, write will remove it and return an initial mouth shape to the user. Note that if you are going to be doing reads, the mouths parameter must be set to 1.

INPUTS

Narrator IORB

ch\_masks - array of audio channel selection masks  
(see audio device documentation for description of this field)  
nm\_masks - number of audio channel selection masks  
mouths - 0 if no mouths are desired  
          1 if mouths are to be read  
rate - speaking rate  
pitch - pitch  
mode - pitch mode  
          0 if natural mode  
          1 if robotic mode  
sex - 0 if male  
      1 if female  
io\_Message - message port  
io\_Command - CMD\_WRITE  
io\_Data - input string  
io\_Length - length of input string

RESULTS

The function sets the io\_Error field of the IORB. The io\_Actual field is set to the length of the input string that was actually processed. If the return code indicates a phoneme error (ND\_PhonErr), io\_Actual is the position in the input string where the error occurred.

SEE ALSO

Read command.  
Audio device documentation.

Contents

parallel.device/AbortIO  
parallel.device/BeginIO  
parallel.device/Clear  
parallel.device/CloseDevice  
parallel.device/Flush  
parallel.device/OpenDevice  
parallel.device/Query  
parallel.device/Read  
parallel.device/Reset  
parallel.device/SetParams  
parallel.device/Start  
parallel.device/Stop  
parallel.device/Write

parallel.device/AbortIO

NAME

AbortIO -- abort the specified I/O request

FUNCTION

This function aborts the specified read or write request. If the request is active, it is stopped immediately. If the request is queued, it is painlessly removed.

INPUTS

ioRequest -- pointer to the IORqst Block that is to be aborted.

RESULTS

Error -- if the Abort succeeded, Error will be #IOERR\_ABORTED (-2) and the request will be flagged as aborted (bit 5 of io\_Flags set). If the Abort failed, the Error will be zero.

parallel.device/BeginIO

NAME

BeginIO -- start up an I/O process

FUNCTION

This function initiates a I/O request made to the parallel device. Other than read or write, the functions are performed synchronously and do not depend on any interrupt handling logic (or its associated discontinuities). If so selected, the function can be performed as IO\_QUICK. Reads and writes are merely initiated by BeginIO, and thus return to the caller as begun, not completed. Completion is signaled via the standard ReplyMsg routine. A valid read or write request is performed asynchronously, never as IO\_QUICK. Multiple requests are handled via FIFO queuing.

INPUTS

ioRequest -- pointer to an I/O Request Block of size io\_ExtParSize (see parallel.i for size/definition), containing a valid function in io\_Command to process, as well as the function's other required parameters.  
deviceNode -- pointer to the "parallel.device" node built at init, and put into io\_Device at Open.

RESULTS

Error -- if the BeginIO succeeded, Error will be null.  
If the BeginIO failed, the Error will be non-zero.  
Most I/O errors won't be reported until the ReplyMsg.

parallel.device/Clear

NAME

Clear -- clear the parallel port buffer

FUNCTION

This function just RTS's (no buffer to clear).

IO REQUEST

|            |                          |
|------------|--------------------------|
| io_Message | mn_ReplyPort initialized |
| io_Device  | set by OpenDevice        |
| io_Unit    | set by OpenDevice        |
| io_Command | CMD_CLEAR                |

RESULTS

Error -- none

parallel.device/Close

NAME

Close -- close the parallel port

SYNOPSIS

CloseDevice(deviceNode)

FUNCTION

This function closes software access to the parallel device.

INPUTS

deviceNode - pointer to the device node, set by Open

SEE ALSO

parallel.device/Open

parallel.device/Flush

NAME

Flush -- clear all queued I/O requests for the parallel port

FUNCTION

This function purges the read and write request queues for the parallel device.

IO REQUEST

io\_Message      mn\_ReplyPort initialized  
io\_Device        set by OpenDevice  
io\_Unit          set by OpenDevice  
io\_Command      CMD\_FLUSH

RESULTS

Error -- if the Flush succeeded, Error will be null.  
          If the Flush failed, the Error will be non-zero.

parallel.device/Open

NAME

Open -- a request to open the parallel port

SYNOPSIS

OpenDevice(parname, unit, ioRequest, flags)

FUNCTION

This function allows the requester software access to the parallel device. Unless the shared-access bit (bit 5 of io\_ParFlags) is set, exclusive use is granted and no other access is allowed until the owner closes the device.

OpenDevice initializes the io\_Device and io\_Unit fields to 0, because there is only one parallel device/unit.

INPUTS

parname    - pointer to literal string "parallel.device"  
unit        - ignored  
ioRequest  - pointer to an ioRequest block of size io\_ExtParSize  
            (see parallel.i for size/definition) to be initialized  
            by the Open routine.  
            NOTE use of io\_ParFlags (see FUNCTION above)

IMPORTANT !!! ioRequest block MUST (!!!) be of size io\_ExtParSize !!!

flags      - ignored

RESULTS

DO -- pointer to the device node  
Error -- if the Open succeeded, Error will be null.  
          If the Open failed, then the Error will be non-zero.



parallel.device/Query

NAME

Query -- query parallel port/line status

FUNCTION

This function return the status of the parallel port lines and registers.

IO REQUEST

|            |                          |
|------------|--------------------------|
| io_Message | mn_ReplyPort initialized |
| io_Device  | set by OpenDevice        |
| io_Unit    | set by OpenDevice        |
| io_Command | PDCMD_QUERY (0A)         |

RESULTS

| io_Status | BIT | ACTIVE | FUNCTION               |
|-----------|-----|--------|------------------------|
|           | 0   | low    | printer selected       |
|           | 1   | low    | paper out              |
|           | 2   | low    | printer in busy toggle |
|           | 3   | -      | read=0,write=1         |
|           | 4-7 |        | reserved               |

parallel.device/Read

NAME

Read -- read input from parallel port

FUNCTION

This function causes a stream of characters to be read from the parallel I/O register. The number of characters is specified in io\_Length, unless -1 is used, in which case input is read until an EOF is read (currently 0x00). If no read request has been made, pending input (i.e. handshake request) is not acknowledged.

IO REQUEST

|            |                                                                            |
|------------|----------------------------------------------------------------------------|
| io_Message | mn_ReplyPort initialized                                                   |
| io_Device  | set by OpenDevice                                                          |
| io_Unit    | set by OpenDevice                                                          |
| io_Command | CMD_READ                                                                   |
| io_Flags   | IOF_QUICK if quick I/O possible and desired                                |
| io_Length  | number of characters to receive, or if set to -1 receive until EOF read in |
| io_Data    | pointer where to put the data.                                             |

RESULTS

Error -- if the Read succeeded, Error will be null. If the Read failed, the Error will be non-zero.

SEE ALSO

parallel.device/BeginIO, parallel.device/SetParams

parallel.device/Reset

NAME

Reset -- reinitialize the parallel port

FUNCTION

This function resets the parallel port to its freshly initialized condition. It aborts all I/O requests both queued and current and sets the port's flags and parameters to their boot-up time default values.

IO REQUEST

|            |                          |
|------------|--------------------------|
| io_Message | mn_ReplyPort initialized |
| io_Device  | set by OpenDevice        |
| io_Unit    | set by OpenDevice        |
| io_Command | CMD_RESET                |

RESULTS

Error -- if the Reset succeeded, Error will be null.  
If the Reset failed, the Error will be non-zero.

parallel.device/SetParams

NAME

SetParams -- change parameters for the parallel port

FUNCTION

This function allows the caller to change parameters for the parallel port. It will disallow changes if any reads or writes are active or queued. The EofMode bit of io\_SerFlags can be set/reset without a call to Setparams. The Shared bit of io\_SerFlags pertains to OpenDevice calls only. ALL OTHER PARAMETERS CAN BE CHANGED ONLY BY THE SETPARAMS FUNCTION. (!!!!)

IO REQUEST

|            |                          |
|------------|--------------------------|
| io_Message | mn_ReplyPort initialized |
| io_Device  | set by OpenDevice        |
| io_Unit    | set by OpenDevice        |
| io_Command | PDCMD_SETPARAMS (09)     |

NOTE: the following fields are filled by Open to reflect the parallel device's current configuration.

|              |                                            |
|--------------|--------------------------------------------|
| io_PExtFlags | not used in V1.1 (MUST be set to zero)     |
| io_ParFlags  | see definition in parallel.i or parallel.h |

NOTE: x00 yields exclusive access, termarray inactive.

|               |                                                                                                                                                                                                                                                                                               |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| io_PTermArray | ASCII descending-ordered 8-byte array of termination characters. If less than 8 chars used, fill out array w/lowest valid value. Terminators are used only if EOFMODE bit of io_Parflags is set. (e.g. x512F040303030303 ) This field is filled on OpenDevice only if the EOFMODE bit is set. |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

RESULTS

Error -- if the SetParams succeeded, Error will be null.  
If the SetParams failed, the Error will be non-zero.

parallel.device/Start

NAME  
Start -- restart I/O that has paused on the parallel port

FUNCTION  
This function restarts the current I/O activity on the parallel port by reactivating the handshaking sequence.

IO REQUEST  
io\_Message mn\_ReplyPort initialized  
io\_Device set by OpenDevice  
io\_Unit set by OpenDevice  
io\_Command CMD\_START

RESULTS  
Error -- if the Start succeeded, Error will be null.  
If the Start failed, the Error will be non-zero.

SEE ALSO  
parallel.device/Stop

parallel.device/Stop

NAME  
Stop -- pause current activity on the parallel port

FUNCTION  
This function halts the current I/O activity on the parallel device by discontinuing the handshaking sequence.

IO REQUEST  
io\_Message mn\_ReplyPort initialized  
io\_Device set by OpenDevice  
io\_Unit set by OpenDevice  
io\_Command CMD\_STOP

RESULTS  
Error -- if the Stop succeeded, Error will be null.  
If the Stop failed, the Error will be non-zero.

SEE ALSO  
parallel.device/Start

parallel.device/Write

NAME

Write -- send output to parallel port

FUNCTION

This function causes a stream of characters to be written to the parallel output register. The number of characters is specified in io\_Length, unless -1 is used, in which case output is sent until an EOF is encountered (currently 0x00).

IO REQUEST

io\_Message      mn\_ReplyPort initialized  
io\_Device        set by OpenDevice  
io\_Unit          set by OpenDevice  
io\_Command      CMD\_WRITE  
io\_Flags         IOF\_QUICK if quick I/O is possible and desired  
io\_Length        number of characters to transmit, or if set  
                  to -1 send until EOF encountered  
io\_Data          pointer to block of data to transmit

RESULTS

Error -- if the Write succeeded, Error will be null.  
          If the Write failed, the Error will be non-zero.

SEE ALSO

parallel.device/BeginIO, parallel.device/SetParams

Contents

printer.device/DumpRPort  
printer.device/Flush  
printer.device/Invalid  
printer.device/PrtCommand  
printer.device/RawWrite  
printer.device/Reset  
printer.device/Start  
printer.device/Stop  
printer.device/Write

printer.device/DumpRPort

NAME

DumpRPort - dump the specified RastPort to a graphics printer

FUNCTION

Prints a rendition of the supplied RastPort, using the supplied ColorMap, position and scaling information, as specified in the printer Preferences

IO REQUEST

io\_Message mn\_ReplyPort set if quick I/O is not possible  
io\_Command PRD\_DUMPRPORT  
io\_Flags IOB\_QUICK set if quick I/O is possible  
io\_RastPort ptr to a RastPort.  
io\_ColorMap ptr to a ColorMap.  
io\_Modes the 'modes' flag as from a ViewPort structure the upper word is reserved and should be zero  
io\_SrcX the x offset into the RastPort  
io\_SrcY the y offset into the RastPort  
io\_SrcWidth the x size in the RastPort to be printed  
io\_SrcHeight the y size in the RastPort to be printed  
io\_DestCols ...  
io\_DestRows these two parameters describe the size of the area to print to on the printer, as described below.

io\_Special a) interpretation of Dest parameters:  
If SPECIAL\_MIL is set, then the associated parameter is specified in thousandths of an inch on the printer.  
If SPECIAL\_FULL is set, then the dimension is set to the maximum possible (as determined by the printer limits or the configuration limits, whichever is less).  
If SPECIAL\_FRAC is set, the parameter is taken to be a longword binary fraction of the maximum for that dimension.  
If ASPECT is set, one of the dimensions may be reduced to preserve the aspect ratio of the print.  
If all bits for a dimension are clear, the parameter is specified in printer pixels.  
If SPECIAL\_DENSITY(1-4) is set, the printer-specific driver has the option of selecting a different dots per inch density (dpi) than the default one. As of this writing, the printer-specific modules supporting this feature are the HP\_LASERJET and the HP\_LASERJET\_PLUS. For these two printers, the densities are 75, 100, 150 & 300 dpi, respectively. The HP\_LASERJET always defaults to 75 dpi. The HP\_LASERJET\_PLUS defaults to 100 dpi if the preferences is set to DRAFT quality and 150 dpi with LETTER quality.  
if SPECIAL\_CENTER is set, then the picture will be centered on the paper.

io\_DestCols that may produce unexpected results when they are not greater than zero and io\_Special is zero. They have been retained for compatability. The user will not trigger these other rules with well formed usage of io\_Special.

The special rules for io\_DestRows and io\_DestCols (WHICH TAKE EFFECT ONLY IF IO\_SPECIAL IS 0) are:

- a) DestCols>0 & DestRows>0 - use as absolute values. i.e., DestCols=320 & DestRows=200 means that the picture will appear on the printer as 320x200 dots.
- b) DestCols=0 & DestRows>0 - use the printer's maximum number of columns and print DestRows lines, i.e., if DestCols=0 and DestRows=200 than the picture will appear on the printer as wide as it can be and 200 dots high.
- c) DestCols=0 & DestRows=0 - same as above except the driver determines the proper number of lines to print based on the aspect ratio of the printer. This results in the largest picture possible that is not distorted or inverted  
Note: As of this writing, this is the call made by such program as DeluxePaint, GraphicCraft, and AegisImages.
- d) DestCols>0 & DestRows=0 - use the specified width and the driver determines the proper number of lines to print based on the aspect ratio of the printer, i.e., if you desire a picture that is 500 pixels wide and aspect ratio correct, use DestCols=500 and DestRows=0.
- e) DestCols<0 or DestRows>0 - the final picture is either a reduction or expansion based on the fraction  $|DestCols| / DestRows$  in the proper aspect ratio.  
Some examples:  
1) if DestCols=-2 & DestRows=1 then the printed picture will be 2x the AMIGA picture and in the proper aspect ratio. ( $2x$  is derived from  $|-2| / 1$  which gives 2.0)  
2) if DestCols=-1 & DestRows=2 then the printed picture will be 1/2x the AMIGA picture in the proper aspect ratio. ( $1/2x$  is derived from  $|-1| / 2$  which gives 0.5)

HINTS

The printer selected in preferences must have graphics capability to use this command.  
Color printers may not be able to print black and white or grey-scale pictures -- specifically, the Okimate 20 cannot print these with a color ribbon: use a black one.  
If the printer has an input buffer option, use it.  
If the printer can be uni- or bidirectional, select unidirectional; this produces a much cleaner picture and in some cases a faster printout.  
Please note that the width and height of the printable area on the printer is in terms of pixels and bounded by the following:  
a) WIDTH = (RIGHT\_MARGIN - LEFT\_MARGIN + 1) / CHARACTERS\_PER\_INCH  
b) HEIGHT = LENGTH / LINES\_PER\_INCH  
For RGB printer support, the YMC values in the printer-specific render.c functions equate to RGB respectively, i.e., yellow is red, magenta is green, and cyan is blue.

There exist rules for the interpretation of io\_DestRows and

printer.device/Flush

NAME

Flush - abort all I/O requests (immediate)

FUNCTION

Flush aborts all stopped I/O at the unit.

IO REQUEST

|            |                                               |
|------------|-----------------------------------------------|
| io_Message | mn_ReplyPort set if quick I/O is not possible |
| io_Device  | preset by the call to OpenDevice              |
| io_Command | CMD_FLUSH                                     |
| io_Flags   | IOB_QUICK set if quick I/O is possible        |

printer.device/Invalid

NAME

Invalid - invalid command

FUNCTION

Invalid is always an invalid command and sets the device error appropriately.

IO REQUEST

|            |                                               |
|------------|-----------------------------------------------|
| io_Message | mn_ReplyPort set if quick I/O is not possible |
| io_Command | CMD_INVALID                                   |
| io_Flags   | IOB_QUICK set if quick I/O is possible        |

printer.device/PrtCommand

NAME

PCPrtCommand -- send a command to the printer.

FUNCTION

This function sends a command to either the parallel or serial device. The printer device maps this command to the control code set of the current printer. The commands supported can be found with the printer.device/Write command. All printers may not support all functions.

IO REQUEST IOPrtCmdReq

|               |                           |
|---------------|---------------------------|
| io_Message    | mn_ReplyPort set          |
| io_Device     | preset by OpenDevice      |
| io_Unit       | preset by OpenDevice      |
| io_Command    | PRD_PRTCOMMAND            |
| io_PrtCommand | the actual command number |
| io_Parm0      | parameter for the command |
| io_Parm1      | parameter for the command |
| io_Parm2      | parameter for the command |
| io_Parm3      | parameter for the command |

RESULTS

Errors: if the PCPrtCommand succeeded, Error will be zero.  
Otherwise, the Error will be non-zero.

SEE ALSO

printer.device/Write printer.h, parallel.device, Preferences

printer.device/RawWrite

NAME

RawWrite - transparent write command

FUNCTION

This is a nonstandard write command that performs no processing on the data passed to it.

IO REQUEST

|            |                                               |
|------------|-----------------------------------------------|
| io_Message | mn_ReplyPort set if quick I/O is not possible |
| io_Command | PRD_RAWWRITE                                  |
| io_Flags   | IOB_QUICK set if quick I/O is possible        |
| io_Length  | the number of bytes in io_Data                |
| io_Data    | the raw bytes to write to the printer         |

printer.device/Reset

NAME

Reset - reset the printer

FUNCTION

Reset resets the printer device without destroying handles to the open device.

IO REQUEST

|            |                                               |
|------------|-----------------------------------------------|
| io_Message | mn_ReplyPort set if quick I/O is not possible |
| io_Device  | preset by the call to OpenDevice              |
| io_Command | CMD_RESET                                     |
| io_Flags   | IOB_QUICK set if quick I/O is possible        |

printer.device/Start

NAME

Start - restart after stop (immediate)

FUNCTION

Start restarts the unit after a stop command.

IO REQUEST

|            |                                               |
|------------|-----------------------------------------------|
| io_Message | mn_ReplyPort set if quick I/O is not possible |
| io_Device  | preset by the call to OpenDevice              |
| io_Command | CMD_START                                     |
| io_Flags   | IOB_QUICK set if quick I/O is possible        |



printer.device/Stop

NAME

Stop - pause current and queued I/O requests (immediate)

FUNCTION

Stop pauses all queued requests for the unit and tries to pause the current I/O request. The only commands that will be allowed to be performed subsequently are immediate I/O requests, which include those to start, flush, and finish the I/O after the stop command.

IO REQUEST

|            |                                               |
|------------|-----------------------------------------------|
| io_Message | mn_ReplyPort set if quick I/O is not possible |
| io_Device  | preset by the call to OpenDevice              |
| io_Command | CMD_STOP                                      |
| io_Flags   | IOB_QUICK set if quick I/O is possible        |

printer.device/Write

NAME

PCWrite -- send output to the printer

FUNCTION

This function causes a buffer of characters to be written to the either the parallel or serial device. The number of characters is specified in io\_Length, unless -1 is used, in which case output is sent until a 0x00 is encountered. The printer device, like the console device, maps ANSI X3.64 style 7-bit printer control codes to the control code set of the current printer. The ANSI codes supported can be found below. All printers may not support all functions.

IO REQUEST

|            |                                                                          |
|------------|--------------------------------------------------------------------------|
| io_Message | mn_ReplyPort set                                                         |
| io_Device  | preset by OpenDevice                                                     |
| io_Unit    | preset by OpenDevice                                                     |
| io_Command | CMD_WRITE                                                                |
| io_Length  | number of characters to process, or if -1, process until EOF encountered |
| io_Data    | pointer to block of data to process                                      |

RESULTS

Errors: if the PCWrite succeeded, Error will be zero. Otherwise, the Error will be non-zero.

SEE ALSO

printer.h, parallel.device, serial.device, Preferences

ANSI X3.64 style COMMANDS

|         |          |                      |
|---------|----------|----------------------|
| aRIS    | ESCc     | reset                |
| aRIN    | ESC#1    | initialize           |
| aIND    | ESCD     | lf                   |
| aNEL    | ESCE     | return,lf            |
| aRI     | ESCM     | reverse lf           |
| aSGR0   | ESC[0m   | normal char set      |
| aSGR3   | ESC[3m   | italics on           |
| aSGR23  | ESC[23m  | italics off          |
| aSGR4   | ESC[4m   | underline on         |
| aSGR24  | ESC[24m  | underline off        |
| aSGR1   | ESC[1m   | boldface on          |
| aSGR22  | ESC[22m  | boldface off         |
| aSFC    | SGR30-39 | set foreground color |
| aSBC    | SGR40-49 | set background color |
| aSHORP0 | ESC[0w   | normal pitch         |
| aSHORP2 | ESC[2w   | elite on             |
| aSHORP1 | ESC[1w   | elite off            |
| aSHORP4 | ESC[4w   | condensed fine on    |
| aSHORP3 | ESC[3w   | condensed off        |
| aSHORP6 | ESC[6w   | enlarged on          |
| aSHORP5 | ESC[5w   | enlarged off         |

aTBSALL  
aEXTEND

ESC#5  
ESC[Pn"x

Set default tabs  
extended commands

|         |              |                         |
|---------|--------------|-------------------------|
| aDEN6   | ESC[6"z      | shadow print on         |
| aDEN5   | ESC[5"z      | shadow print off        |
| aDEN4   | ESC[4"z      | doublestrike on         |
| aDEN3   | ESC[3"z      | doublestrike off        |
| aDEN2   | ESC[2"z      | NIQ on                  |
| aDEN1   | ESC[1"z      | NIQ off                 |
|         |              |                         |
| aSUS2   | ESC[2v       | superscript on          |
| aSUS1   | ESC[1v       | superscript off         |
| aSUS4   | ESC[4v       | subscript on            |
| aSUS3   | ESC[3v       | subscript off           |
| aSUS0   | ESC[0v       | normalize the line      |
| aPLU    | ESCL         | partial line up         |
| aPLD    | ESCK         | partial line down       |
|         |              |                         |
| aFNT0   | ESC(B        | US char set             |
| aFNT1   | ESC(R        | French char set         |
| aFNT2   | ESC(K        | German char set         |
| aFNT3   | ESC(A        | UK char set             |
| aFNT4   | ESC(E        | Danish I char set       |
| aFNT5   | ESC(H        | Sweden char set         |
| aFNT6   | ESC(Y        | Italian char set        |
| aFNT7   | ESC(Z        | Spanish char set        |
| aFNT8   | ESC(J        | Japanese char set       |
| aFNT9   | ESC(6        | Norwegian char set      |
| aFNT10  | ESC(C        | Danish II char set      |
|         |              |                         |
| aPROP2  | ESC[2p       | proportional on         |
| aPROP1  | ESC[1p       | proportional off        |
| aPROP0  | ESC[0p       | proportional clear      |
| aTSS    | ESC[n E      | set proportional offset |
| aJFY5   | ESC[5 F      | auto left justify       |
| aJFY7   | ESC[7 F      | auto right justify      |
| aJFY6   | ESC[6 F      | auto full justify       |
| aJFY0   | ESC[0 F      | auto justify off        |
| aJFY3   | ESC[3 F      | letter space (justify)  |
| aJFY1   | ESC[1 F      | word fill(auto center)  |
|         |              |                         |
| aVERP0  | ESC[0z       | 1/8" line spacing       |
| aVERP1  | ESC[1z       | 1/6" line spacing       |
| aSLPP   | ESC[nt       | set form length n       |
| aPERF   | ESC[nq       | perf skip n (n>0)       |
| aPERF0  | ESC[0q       | perf skip off           |
|         |              |                         |
| aLMS    | ESC#9        | Left margin set         |
| aRMS    | ESC#0        | Right margin set        |
| aTMS    | ESC#8        | Top margin set          |
| aBMS    | ESC#2        | Bottom marg set         |
| aSTBM   | ESC[Pn1;Pn2r | T&B margins             |
| aSLRM   | ESC[Pn1;Pn2s | L&R margin              |
| aCAM    | ESC#3        | Clear margins           |
|         |              |                         |
| aHTS    | ESCH         | Set horiz tab           |
| aVTS    | ESCJ         | Set vertical tabs       |
| aTBC0   | ESC[0g       | Clr horiz tab           |
| aTBC3   | ESC[3g       | Clear all h tab         |
| aTBC1   | ESC[1g       | Clr vertical tabs       |
| aTBC4   | ESC[4g       | Clr all v tabs          |
| aTBCALL | ESC#4        | Clr all h & v tabs      |

Contents

serial.device/AbortIO  
serial.device/BeginIO  
serial.device/Break  
serial.device/Clear  
serial.device/Close  
serial.device/Flush  
serial.device/Open  
serial.device/Query  
serial.device/Read  
serial.device/Reset  
serial.device/SetParams  
serial.device/Start  
serial.device/Stop  
serial.device/Write

serial.device/AbortIO

NAME

AbortIO -- abort the specified I/O request

FUNCTION

This function aborts the specified read or write request. If the request is active, it is stopped immediately. If the request is queued, it is painlessly removed.

INPUTS

ioRequest -- pointer to the IORqst Block that is to be aborted.

RESULTS

Error -- if the Abort succeeded, Error will be #IOERR\_ABORTED (-2) and the request will be flagged as aborted (set bit 5 of io\_Flags). If the Abort failed, the Error will be zero.

serial.device/BeginIO

NAME

BeginIO -- start up an I/O process

FUNCTION

This function initiates a I/O request made to the serial device. Other than read or write, the functions are performed synchronously and do not depend on any interrupt-handling logic (or its associated discontinuities). Hence, if so selected, the functions can be performed as IO\_QUICK. With one exception, reads and writes are merely initiated by BeginIO and thus return to the caller as begun, not completed. Completion is signaled via the standard ReplyMsg routine. Multiple requests are handled via FIFO queuing. The only exception to this non-QUICK handling of reads and writes is for READS when:

- IO\_QUICK bit is set
- There are no pending read requests
- There is already enough data in the input buffer to satisfy this I/O Request immediately.

In this case, the IO\_QUICK flag is not cleared and the request is completed by the time it returns to the caller. There is no ReplyMsg or signal bit activity in this case.

INPUTS

ioRequest -- pointer to an I/O Request Block of size io\_ExtSerSize (see serial.i for size/definition), containing a valid command in io\_Command to process, as well as the command's other required parameters.  
deviceNode -- pointer to the "serial.device" node built at init, and put into io\_Device at Open.

RESULTS

Error -- if the BeginIO succeeded, Error will be null. If the BeginIO failed, the Error will be non-zero. Most I/O errors won't be reported until the ReplyMsg.

serial.device/Break

NAME

Break -- send a break signal over the serial line

FUNCTION

This function sends a break signal (serial line held low for an extended period) out the serial port. This is accomplished by setting the UARTBRK bit of reg ADKCON. After a duration (user-specifiable via setparams, default 25000 microseconds), the bit is reset and the signal discontinued. If the QUEUEDBRK bit of io\_SerFlags is set in the io\_Request block, the request is placed at the back of the write-request queue and executed in turn. If the QUEUEDBRK bit is not set, the break is started immediately, control returns to the caller, and the timer discontinues the signal after the duration is completed. It is up to the caller to coordinate his/her intentions with the proper commands such as ABORT, FLUSH, STOP, START, etc.

IO REQUEST

|            |                                          |
|------------|------------------------------------------|
| io_Message | mn_ReplyPort initialized                 |
| io_Device  | set by OpenDevice                        |
| io_Unit    | set by OpenDevice                        |
| io_Command | SDCMD_BREAK                              |
| io_Flags   | set/reset IO_QUICK per above description |

RESULTS

Error -- if the Break succeeded, Error will be null. If the Break failed, the Error will be non-zero.

serial.device/Clear

NAME

Clear -- clear the serial port buffers

FUNCTION

This function resets the serial port's read buffer pointers.

IO REQUEST

|            |                          |
|------------|--------------------------|
| io_Message | mn_ReplyPort initialized |
| io_Device  | set by OpenDevice        |
| io_Unit    | set by OpenDevice        |
| io_Command | CMD_CLEAR                |

RESULTS

Error -- if the Clear succeeded, Error will be null.  
If the Clear failed, the Error will be non-zero.

serial.device/Close

NAME

Close -- close the serial port

SYNOPSIS

CloseDevice(deviceNode)

FUNCTION

This function closes software access to the serial device. Upon closing, the device's input buffer is freed.

INPUTS

deviceNode -- pointer the device node, set by Open

SEE ALSO

serial.device/Open

serial.device/Flush

NAME

Flush -- clear all queued I/O requests for the serial port

FUNCTION

This function purges the read and write request queues for the serial device. Flush will not affect active requests.

IO REQUEST

io\_Message      mn\_ReplyPort initialized  
io\_Device        set by OpenDevice  
io\_Unit          set by OpenDevice  
io\_Command      CMD\_FLUSH

RESULTS

Error -- if the Flush succeeded, Error will be null.  
          If the Flush failed, the Error will be non-zero.

serial.device/Open

NAME

Open -- a request to open the serial port

SYNOPSIS

OpenDevice(sername, unit, ioRequest, flags)

FUNCTION

This function allows the requester software access to the serial device. Unless the shared-access bit (bit 5 of io\_SerFlags) is set, exclusive use is granted and no other access is allowed until the owner closes the device. All serial-specific fields are initialized to their most recent values (or default, if the first time open). OpenDevice initializes the io\_Device and io\_Unit fields to 0, since there is only one serial device/unit. If the user wants to support 7-wire handshaking (i.e. RS232-C CTS/RTS protocol), he should set the 7WIRE bit before opening.

INPUTS

sername    - pointer to literal string "serial.device"  
unit        - ignored  
ioRequest  - pointer to an ioRequest block of size io\_ExtSerSize  
            (see serial.i,h for size/definition) to be initialized  
            by the OpenDevice routine.

NOTE use of io\_SerFlags (see FUNCTION above)

##&#! IMPORTANT !!! ioRequest block MUST (!!) be of size io\_ExtSerSize !!!  
flags       - ignored

RESULTS

DO -- pointer to the device node  
Error -- if the Open succeeded, Error will be null.  
          If the Open failed, the Error will be non-zero.

serial.device/Query

NAME

Query -- query serial port/line status

FUNCTION

This function returns the status of the serial port lines and registers. The number of unread bytes in the serial device's read buffer is shown in io\_Actual.

IO REQUEST

io\_Message mn\_ReplyPort initialized  
 io\_Device set by OpenDevice  
 io\_Unit set by OpenDevice  
 io\_Command SDCMD\_QUERY (0A)

RESULTS

| io_Status | BIT   | ACTIVE | FUNCTION                         |
|-----------|-------|--------|----------------------------------|
| LSB       | 0     | low    | reserved                         |
|           | 1     | low    | reserved                         |
|           | 2     | low    | reserved                         |
|           | 3     | low    | Data Set Ready                   |
|           | 4     | low    | Clear To Send                    |
|           | 5     | low    | Carrier Detect                   |
|           | 6     | low    | Ready To Send                    |
| MSB       | 7     | low    | Data Terminal Ready              |
|           | 8     | high   | read buffer overflow             |
|           | 9     | high   | break sent (most recent output)  |
|           | 10    | high   | break received (as latest input) |
|           | 11    | high   | transmit x-OFFed                 |
|           | 12    | high   | receive x-OFFed                  |
|           | 13-15 |        | reserved                         |

io\_Actual set to count of unread input characters

Error -- if the Query succeeded, Error will be null.

If the Flush failed, the Error will be non-zero.

serial.device/Read

NAME

Read -- read input from serial port

FUNCTION

This function causes a stream of characters to be read in the serial port. The number of characters is specified in io\_Length, unless -1 is used, in which case input is read until a null(0x00) is received. Input for which there is no request is stored in the input buffer until it can be dispatched to a requester.

IO REQUEST

io\_Message mn\_ReplyPort initialized  
 io\_Device set by OpenDevice  
 io\_Unit set by OpenDevice  
 io\_Command CMD\_READ

io\_Flags IOF\_QUICK if quick I/O possible and desired  
 io\_Length number of characters to receive, or if set to -1 receive until null(0x00) read in  
 io\_Data pointer to read buffer

RESULTS

Error -- if the Read succeeded, Error will be null.  
If the Read failed, the Error will be non-zero.

serial.device/Reset

NAME

Reset -- reinitialize the serial port

FUNCTION

This function resets the serial port to its freshly initialized condition. It aborts all I/O requests both queued and current, relinquishes the current buffer, obtains a new default sized buffer, and sets the port's flags and parameters to their boot-up time default values. The function places the reset parameter values in the ioRequest block.

IO REQUEST

|            |                          |
|------------|--------------------------|
| io_Message | mn_ReplyPort initialized |
| io_Device  | set by OpenDevice        |
| io_Unit    | set by OpenDevice        |
| io_Command | CMD_RESET                |

RESULTS

Error -- if the Reset succeeded, Error will be null.  
 If the Reset failed, the Error will be non-zero.

serial.device/SetParams

NAME

SetParams -- change parameters for the serial port

FUNCTION

This function allows the caller to change parameters for the serial device. Except for xON-xOFF enable/disable, it will reject a setparams call if any reads or writes are active or pending.

Note specifically:

1. Valid input for io\_Baud is between 112 and 292000 baud inclusive; asynchronous I/O above 32KB (especially on a busy system) may be ambitious.
2. The EOFMODE and QUEUEDBRK bits of io\_SerFlags can be set/reset in the io\_Rqst block without a call to SetParams. The SHARED and 7WIRE bits of io\_SerFlags are used in OpenDevice calls. ALL OTHER PARAMETERS CAN BE CHANGED ONLY BY THE SetParams COMMAND. (!!!!)
3. RBufLen must be at least 512.
4. io\_ExtFlags is not used in V1.1 and MUST be set to zero to assure upward compatibility.
5. xON-xOFF is by default enabled. The XDISABLED bit is the only parameter that can be changed via a SetParams call while the device is active. Note that this will return the value SerErr\_DevBusy in the io\_Error field.
6. If you are trying to run MIDI, it is suggested to set the RAD\_BOOGIE bit of io\_SerFlags to bypass unneeded overhead. Specifically, this skips checks for parity, x-OFF handling, character lengths other than 8 bits, and testing for a break signal. Setting RAD\_BOOGIE will also set the XDISABLED bit.  
 Note that writing data (that's already in MIDI format) at MIDI rates is easily accomplished. Using this driver alone for MIDI reads may, however, be inappropriate, because of MIDI time-stamping requirements and the possibility of overruns in a busy multitasking and/or display-intensive environment.

IO REQUEST

|            |                          |
|------------|--------------------------|
| io_Message | mn_ReplyPort initialized |
| io_Device  | set by OpenDevice        |
| io_Unit    | set by OpenDevice        |
| io_Command | SDCMD_SETPARAMS (0x0B)   |

NOTE: the following fields are filled in by Open to reflect the serial device's current configuration.

|             |                                                                                                                  |
|-------------|------------------------------------------------------------------------------------------------------------------|
| io_CtlChar  | a longword containing byte values for the xON,xOFF,INQ,ACK fields (respectively) (INQ/ACK not used at this time) |
| io_RBufLen  | length in bytes of input buffer                                                                                  |
| io_ExtFlags | (not used)                                                                                                       |

NOTE: any change in buffer size causes the current buffer to be deallocated and a new, correctly sized one to be allocated. Thus, the CONTENTS OF THE OLD BUFFER ARE LOST.

|              |                                               |
|--------------|-----------------------------------------------|
| io_Baud      | baud rate for reads AND writes. (See 1 above) |
| io_BrkTime   | duration of break signal in MICROseconds      |
| io_TermArray | ASCII descending-ordered 8-byte array of      |



termination characters. If less than 8 chars  
 used, fill out array w/lowest valid value.  
 Terminators are checked only if EOFMODE bit of  
 io\_SerFlags is set. (e.g., x512F040303030303 )  
 io\_ReadLen number of bits in read word (1-8) not including parity  
 io\_WriteLen number of bits in write word (1-8) " " "  
 io\_StopBits number of stop bits (1 normal, 2 can be  
 specified for reads if ReadLen <= 7)  
 io\_SerFlags see serial.i,h for bit equates, NOTE that x00  
 yields exclusive access, xON/OFF-enabled, no  
 parity checking, 3-wire protocol and TermArray  
 inactive.

RESULTS

Error -- if the SetParams succeeded, Error will be null.  
 If the SetParams failed, the Error will be non-zero.

serial.device/Start

NAME

Start -- restart paused I/O over the serial port

FUNCTION

This function restarts all current I/O on the serial port by  
 sending an xON to the "other side," and submitting a "logical  
 xON" to "our side," if/when appropriate to current activity.

IO REQUEST

|            |                          |
|------------|--------------------------|
| io_Message | mn_ReplyPort initialized |
| io_Device  | set by OpenDevice        |
| io_Unit    | set by OpenDevice        |
| io_Command | CMD_START                |

RESULTS

Error -- if the Start succeeded, Error will be null.  
 If the Start failed, the Error will be non-zero.

SEE ALSO

serial.device/Stop

serial.device/Stop

NAME

Stop -- pause all current I/O over the serial port

FUNCTION

This function halts all current I/O on the serial port by sending an xOFF to the "other side," and submitting a "logical xOFF" to "our side," if/when appropriate to current activity.

IO REQUEST

|            |                          |
|------------|--------------------------|
| io_Message | mn_ReplyPort initialized |
| io_Device  | set by OpenDevice        |
| io_Unit    | set by OpenDevice        |
| io_Command | CMD_STOP                 |

RESULTS

Error -- if the Stop succeeded, Error will be null.  
If the Stop failed, the Error will be non-zero.

SEE ALSO

serial.device/Start

serial.device/Write

NAME

Write -- send output to serial port

FUNCTION

This function causes a stream of characters to be written out the serial port. The number of characters is specified in io\_Length, unless -1 is used, in which case output is sent until a null(0x00) is encountered.

IO REQUEST

|            |                                                                                             |
|------------|---------------------------------------------------------------------------------------------|
| io_Message | mn_ReplyPort initialized                                                                    |
| io_Device  | set by OpenDevice                                                                           |
| io_Unit    | set by OpenDevice                                                                           |
| io_Command | CMD_WRITE                                                                                   |
| io_Flags   | IOF_QUICK set if quick I/O possible and desired                                             |
| io_Length  | number of characters to transmit, or if set to -1 transmit until null encountered in buffer |
| io_Data    | pointer to block of data to transmit                                                        |

RESULTS

Error -- if the Write succeeded, Error will be null.  
If the Write failed, the Error will be non-zero.

SEE ALSO

serial.device/BeginIO, serial.device/setParams

Contents

timer.device/AddTime  
timer.device/background  
timer.device/CmpTime  
timer.device/SubTime  
timer.device/TR\_ADDREQUEST  
timer.device/TR\_GETSYSTIME  
timer.device/TR\_SETSYSTIME

timer.device/AddTime

NAME

AddTime -- add one time request to another

SYNOPSIS

AddTime( Dest, Source ), timer.device  
          A0   A1      A6

FUNCTION

This routine adds one timeval structure to another. The results are stored in the destination (Dest + Source -> Dest)

A0 and A1 will be left unchanged.

INPUTS

Dest, Source -- pointers to timeval structures.

EXCEPTIONS

SEE ALSO

BUGS

timer.device/background

#### TIMER REQUEST

A time request is a nonstandard IO Request. It has an IORequest followed by a timeval structure.

#### TIMEVAL

A timeval structure consists of two longwords. The first is the number of seconds, the latter is the fractional number of microseconds. The microseconds must always be "normalized;" e.g., the longword must be between 0 and one million.

#### UNITS

The timer contains two units -- one that is precise but inaccurate, the other that has little system overhead, is very stable over time, but has only limited resolution.

#### UNIT\_MICROHZ

This unit uses a programmable timer in the 8520 to keep track of its time. It has precision down to about 2 microseconds, but will drift as system load increases. The timer is typically accurate to within five percent.

#### UNIT\_VBLANK

This unit is driven by the vertical blank interrupt. It is very stable over time, but has a resolution of only 16667 microseconds (or 20000 microseconds in PAL land). The timer is cheap to use, and should be used by those who are waiting for long periods of time (typically 1/2 second or more).

#### LIBRARY

In addition to the normal device calls, the timer also supports three direct, library-like calls. They are for manipulating timeval structures. Addition, subtraction, and comparison are supported.

timer.device/CmpTime

#### NAME

CmpTime - compare two timeval structures

#### SYNOPSIS

```
result = CmpTime(Dest, Source), timer.device
 A0 A1 A6
```

#### FUNCTION

This routine compares two timeval structures.

A0 and A1 will be left unchanged.

#### INPUTS

Dest, Source -- pointers to timeval structures.

#### RESULTS

```
result = 0 if Dest has the same time as Source
result = -1 if Dest has less time than Source
result = +1 if Dest has more time than Source
```

#### EXCEPTIONS

#### SEE ALSO

#### BUGS

timer.device/SubTime

NAME

SubTime - subtract one time request from another

SYNOPSIS

SubTime( Dest, Source ), timer.device  
A0 A1 A6

FUNCTION

This routine subtracts one timeval structure from another. The results are stored in the destination (Dest - Source -> Dest)

A0 and A1 will be left unchanged.

INPUTS

Dest, Source -- pointers to timeval structures.

EXCEPTIONS

SEE ALSO

BUGS

timer.device/TR\_ADDREQUEST

NAME

TR\_ADDREQUEST -- submit a request to time time

FUNCTION

Asks the timer to count off a specified amount of time. The timer will chain this request with its other requests and will reply the message back to the user when the timer counts down to zero.

TIMER REQUEST

|            |                                                                  |
|------------|------------------------------------------------------------------|
| io_Message | mn_ReplyPort initialized                                         |
| io_Device  | preset by timer in OpenDevice                                    |
| io_Unit    | preset by timer in OpenDevice                                    |
| io_Command | TR_ADDREQUEST                                                    |
| io_Flags   | IOF_QUICK allcvable                                              |
| tr_time    | a timeval structure specify how long until the driver will reply |

RESULTS

|         |                   |
|---------|-------------------|
| tr_time | will contain junk |
|---------|-------------------|

timer.device/TR\_GETSYSTIME

NAME

TR\_GETSYSTIME -- get the system time

FUNCTION

Asks the timer what time it is. The system time starts at zero at power-on but may be initialized via the TR\_SETSYSTIME call.

System time is monotonically increasing and guaranteed to be unique (except if someone sets the time backwards). The time is incremented every vertical blank by the vertical blanking interval. In addition, it is changed every time someone asks what time it is. In this way, the return value of the system time is unique and unrepeating.

TIMER REQUEST

|            |                               |
|------------|-------------------------------|
| io_Message | mn_ReplyPort initialized      |
| io_Device  | preset by timer in OpenDevice |
| io_Unit    | preset by timer in OpenDevice |
| io_Command | TR_ADDREQUEST                 |
| io_Flags   | IOF_QUICK allowable           |

RESULTS

|         |                                                                      |
|---------|----------------------------------------------------------------------|
| tr_time | the timeval structure will be filled in with the current system time |
|---------|----------------------------------------------------------------------|

timer.device/TR\_SETSYSTIME

NAME

TR\_SETSYSTIME -- set the system time

FUNCTION

Sets the system's idea of what time it is. The system starts out at time "zero" so it is safe to set it forward to the "real" time. However, care should be taken when setting the time backwards. System time is specified as being monotonically increasing.

TIMER REQUEST

|            |                                                  |
|------------|--------------------------------------------------|
| io_Message | mn_ReplyPort initialized                         |
| io_Device  | preset by timer in OpenDevice                    |
| io_Unit    | preset by timer in OpenDevice                    |
| io_Command | TR_ADDREQUEST                                    |
| io_Flags   | IOF_QUICK allowable                              |
| tr_time    | a timeval structure with the current system time |

RESULTS

none



# Appendix C

## Resource Summaries

This appendix contains summaries for system resource routines. Resources are software entities in the Amiga kernel software that enable cooperating tasks to gain exclusive access to certain parts of the Amiga hardware. There are four resources in the Amiga system:

**disk** allows access to one of four possible disk units.

**cia** allows you to access specific bits in each of the Complex Interface Adapters.

There are two cia resources: `ciaa.resource` and `ciab.resource`, corresponding to the first and second 8520 in the system. See the software memory map in *Amiga ROM Kernel Reference Manual: Exec* for the definition of the bits controlled by each cia.



potgo manages the bits of the POTGO register.

misc manages the serial and parallel port register bits.

Each routine for resource management is outlined in the summary sections that follow.

*Note:* Resources need be used only if you are attempting to use the associated hardware directly. The system software routines use these resources internally when they perform hardware operations. Tasks that also use these software resource controls will be compatible with Exec and the system software.

To use the routines listed for the resources, you must first open the resource and assign the value returned to a specific base pointer name. Here is a list of the resource names and their associated base pointer names. Like names for libraries, their names are null-terminated strings:

| Resource Name  | Base Pointer Name                                     |
|----------------|-------------------------------------------------------|
| potgo.resource | PotgoBase                                             |
| disk.resource  | None provided, for assembly-language programmers only |
| misc.resource  | None provided, for assembly-language programmers only |
| ciaa.resource  | <user-defined>                                        |
| ciab.resource  | <user-defined>                                        |

Some examples follow.

```
struct Library *PotgoBase;
PotgoBase = (struct Library *)OpenResource("potgo.resource");
/* then use the routines provided */

....

/* <user-defined> example */
struct Library *myCiaPointerA;

myCiaPointerA = (struct Library *)OpenResource("ciaa.resource");

/* then utilize myCiaPointerA as one of the explicit parameters
 * for the C language calls to the resource routines. */
```

Contents

|                  |                |
|------------------|----------------|
| AbleICR          | cia.resource   |
| AddICRVector     | cia.resource   |
| RemICRVector     | cia.resource   |
| SetICR           | cia.resource   |
| AllocUnit        | disk.resource  |
| FreeUnit         | disk.resource  |
| CetUnit          | disk.resource  |
| CetUnitID        | disk.resource  |
| GiveUnit         | disk.resource  |
| FreeMiscResource | misc.resource  |
| GetMiscResource  | misc.resource  |
| AllocPotBits     | potgo.resource |
| FreePotBits      | potgo.resource |
| WritePotgo       | potgo.resource |

cia.resource/AbleICR

NAME

AbleICR -- enable/disable ICR interrupts

SYNOPSIS

```
oldMask = AbleICR(Resource, mask)
D0 A6 D0
```

FUNCTION

This function provides a means of enabling and disabling 8520 CIA interrupt control registers. In addition, it returns the previous enable mask.

INPUTS

mask - a bit mask indicating which interrupts to be modified. If bit 7 is clear the mask indicates interrupts to be disabled. If bit 7 is set, the mask indicates interrupts to be enabled. Bit positions are identical to those in 8520 ICR.

resource - pointer to ciaa.resource or ciab.resource as obtained from the call to OpenResource

RESULTS

oldMask - the previous enable mask before the requested changes. To get the current mask without making changes, call the function with a null parameter.

EXAMPLES

```
Get the current mask:
mask = AbleICR(0)
Enable both timer interrupts:
AbleICR(0x83)
Disable serial port interrupt:
AbleICR(0x08)
```

EXCEPTIONS

Enabling the mask for a pending interrupt will cause an immediate processor interrupt (that is, if everything else is enabled). You may want to clear the pending interrupts with SetICRx prior to enabling them.

SEE ALSO

SetICR

cia.resource/AddICRVector

NAME

AddICRVector -- attach an interrupt handler to a CIA bit

SYNOPSIS

```
interrupt = AddICRVector(resource, iCRBit, interrupt)
D0 A6 D0 A1
```

FUNCTION

Assign interrupt processing code to a particular interrupt bit of the CIA ICR. If the interrupt bit has already been assigned, this function will fail, and return a pointer to the owner interrupt. If it succeeds, a null is returned.

This function will also enable the CIA interrupt for the given ICR bit.

INPUTS

iCRBit - bit number to set (0..4)  
interrupt - pointer to interrupt structure  
resource - pointer to ciaa.resource or ciab.resource as obtained from the call to OpenResource

RESULT

interrupt - zero if successful, otherwise returns a pointer to the current owner interrupt structure.

SEE ALSO

RemICRVector

cia.resource/RemICRVector

NAME

RemICRVector -- detach an interrupt handler from a CIA bit

SYNOPSIS

```
RemICRVector(resource, iCRBit, interrupt)
A6 D0 A1
```

FUNCTION

Disconnect interrupt processing code for a particular interrupt bit of the CIA ICR.

This function will also disable the CIA interrupt for the given ICR bit.

INPUTS

iCRBit - bit number to set (0..4)  
interrupt - pointer to interrupt structure  
resource - pointer to ciaa.resource or ciab.resource as obtained from the call to OpenResource

RESULT

SEE ALSO

AddICRVector

cia.resource/SetICR

NAME

SetICR -- cause, clear, and sample ICR interrupts

SYNOPSIS

oldMask = SetICR(resource, mask)  
D0                   A6           D0

FUNCTION

This function provides a means of resetting, causing, and sampling 8520 CIA interrupt control registers.

INPUTS

mask - a bit mask indicating which interrupts to be caused. If bit 7 is clear the mask indicates interrupts to be reset. If bit 7 is set, the mask indicates interrupts to be caused. Bit positions are identical to those in 8520 ICR.  
resource - pointer to c1aa.resource or c1ab.resource as obtained from the call to OpenResource

RESULTS

oldMask - the previous interrupt register status before making the requested changes. To sample current status without making changes, call the function with a null parameter.

EXAMPLES

Get the interrupt mask:  
mask = SetICR(0)  
Clear serial port interrupt:  
SetICR(0x08)

EXCEPTIONS

Setting an interrupt bit for an enabled interrupt will cause an immediate interrupt.

SEE ALSO

AbleICR

disk.resource/AllocUnit

NAME

AllocUnit - allocate a unit of the disk

SYNOPSIS

Success = AllocUnit( unitNum ), DRResource  
D0                   D0           A6

FUNCTION

This routine allocates one of the units of the disk. It should be called before trying to use the disk (via GetUnit).

INPUTS

unitNum -- a legal unit number (zero through three)

RESULTS

Success -- nonzero if successful, zero on failure

EXCEPTIONS

SEE ALSO

BUGS

disk.resource/FreeUnit

NAME

FreeUnit - deallocate the disk

SYNOPSIS

FreeUnit( unitNum ), DRRResource  
D0 A6

FUNCTION

This routine deallocates one of the units of the disk. It should be called when done with the disk. Do not call it if you did not successfully allocate the disk (there is no protection -- you will probably crash the disk system).

INPUTS

unitNum -- a legal unit number (zero through three)

RESULTS

EXCEPTIONS

SEE ALSO

BUGS

disk.resource/GetUnit

NAME

GetUnit - allocate the disk for a driver

SYNOPSIS

lastDriver = GetUnit( unitPointer ), DRRResource  
D0 A1 A6

FUNCTION

This routine allocates the disk to a driver. It is either immediately available, or the request is saved until the disk is available. When it is available, your unitPointer is sent back to you (via ReplyMsg). You may then reattempt the GetUnit.

Allocating the disk allows you to use the disk's resources. Remember however that there are four units to the disk; you are only one of them. Please be polite to the other units (by never selecting them, and by not leaving interrupts enabled, etc.).

When you are done, please leave the disk in the following state:

dmacon dma bit ON  
dsklen dma bit OFF (write a #DSKDMAOFF to dsklen)  
adkcon disk bits -- any way you want  
entena:disk sync and disk block interrupts -- Both DISABLED  
CIA resource index interrupt -- DISABLED  
8520 outputs -- doesn't matter, because all bits will be set to inactive by the resource.  
8520 data direction regs -- restore to original state.

INPUTS

unitPtr - a pointer to your disk resource unit structure.  
Note that the message filed of the structure MUST be a valid message, ready to be replied to.

RESULTS

lastDriver - if the disk is not busy, then the last unit to use the disk is returned. This may be used to see if a driver needs to reset device registers. (If you were the last user, then no one has changed any of the registers. If someone else has used it, then any allowable changes may have been made). If the disk is busy, then a null is returned.

EXCEPTIONS

SEE ALSO

BUGS

disk.resource/GetUnitID

NAME

GetUnitID - find out what type of disk is out there

SYNOPSIS

idtype = GetUnitID( unitNum ), DRResource  
D0                    D0                    A6

FUNCTION

INPUTS

RESULTS

idtype -- the type of the disk drive. Standard types are defined in the resource include file.

EXCEPTIONS

SEE ALSO

BUGS

disk.resource/GiveUnit

NAME

GiveUnit - Free the disk

SYNOPSIS

GiveUnit(), DRResource  
A6

FUNCTION

This routine frees the disk after a driver is done with it. If others are waiting, it will notify them.

INPUTS

RESULTS

EXCEPTIONS

SEE ALSO

BUGS

misc.resource/FreeMiscResource

NAME

FreeMiscResource - make a resource available for reallocation

SYNOPSIS

FreeMiscResource( unitNum ), DRResource  
D0 A6

FUNCTION

This routine frees one of the resources allocated by AllocMiscResource. The resource is made available for reuse.

This routine may not be called from an interrupt routine

INPUTS

unitNum - the number of the miscellaneous resource to be freed.

RESULTS

EXCEPTIONS

SEE ALSO

BUGS

misc.resource/GetMiscResource

NAME

GetMiscResource - allocate one of the misc resources

SYNOPSIS

CurrentUser = GetMiscResource( unitNum, name ), DRResource  
D0 D0 A1 A6

FUNCTION

This routine allocates one of the miscellaneous resources. If the resource is currently allocated, an error is returned. If you do get it, your name is associated with the resource (so a user can see who has it allocated).

This routine may not be called from an interrupt routine

INPUTS

unitNum - the number of the resource you want to allocate  
name - a mnemonic name that will help the user figure out what piece of software is hogging a resource.  
(havoc breaks out if a name of null is passed in...)

RESULTS

CurrentUser - if the resource is busy, then the name of the current user is returned. If the resource is free, then null is returned.

EXCEPTIONS

SEE ALSO

BUGS

potgo.resource/AllocPotBits

NAME

AllocPotBits - allocate bits in the potgo register

SYNOPSIS

allocated = AllocPotBits(bits), potgoResource  
D0 D0 A6

FUNCTION

The AllocPotBits routine allocates bits in the hardware potgo register that the application wishes to manipulate via WritePotgo. The request may be for more than one bit. A user trying to allocate bits may find that they are unavailable because they are already allocated, or because the start bit itself (bit 0) has been allocated, or if requesting the start bit, because input bits have been allocated. A user can block itself from allocation: i.e., it should FreePotgoBits the bits it has and re-AllocPotBits if it is trying to change an allocation involving the start bit.

INPUTS

bits - a description of the hardware bits that the application wishes to manipulate, loosely based on the register description itself:

- START (bit 0) - set if you wish to use start (i.e., start the proportional controller counters) with the input ports you allocate (below). You must allocate all the DATxx ports you want to apply START to in this same call, with the OUTxx bit clear.
- DATLX (bit 8) - set if you wish to use the port associated with the left (0) controller, pin 5.
- OUTLX (bit 9) - set if you promise to use the LX port in output mode only. The port is not set to output for you at this time -- this bit set indicates that you don't mind if STARTs are initiated at any time by others, since ports that are enabled for output are unaffected by START.
- DATLY (bit 10) - same as DATLX but for the left (0) controller, pin 9.
- OUTLY (bit 11) - same as OUTLX but for LY.
- DATRX (bit 12) - the right (1) controller, pin 5.
- OUTRX (bit 13) - OUT for RX.
- DATRY (bit 14) - the right (1) controller, pin 9.
- OUTRY (bit 15) - OUT for RY.

RESULTS

allocated - the START and DATxx bits of those requested that were granted. The OUTxx bits are don't cares.

potgo.resource/FreePotBits

NAME

FreePotBits - free allocated bits in the potgo register

SYNOPSIS

FreePotBits(allocated), potgoResource  
D0 A6

FUNCTION

The FreePotBits routine frees previously allocated bits in the hardware potgo register that the application had allocated via AllocPotBits and no longer wishes to use. It accepts the return value from AllocPotBits as its argument.



potgo.resource/WritePotgo

NAME

WritePotgo - write to the hardware potgo register

SYNOPSIS

WritePotgo(word, mask), potgoResource  
D0 D1 A6

FUNCTION

The WritePotgo routine sets and clears bits in the hardware potgo register. Only those bits specified by the mask are affected -- it is improper to set bits in the mask that you have not successfully allocated. The bits in the high byte are saved to be maintained when other users write to the potgo register. The START bit is not saved, it is written only explicitly as the result of a call to this routine with the START bit set: other users will not restart it.

INPUTS

word - the data to write to the hardware potgo register and save for further use, except the START bit, which is not saved.  
mask - those bits in word that are to be written. Other bits may have been provided by previous calls to this routine, and default to zero.

# Appendix D

## Include Files

This appendix has separate sections for the C and assembly-language include files. At the beginning of each section of files there is a cross-reference showing all the defined constants, data structures, and data structure terms in each file. These names are listed alphabetically, followed by file and line-number references.

## **C Include Files — “.h” Files**

The first portion of this appendix contains the C-language include files that define the system data structures used by the ROM (or kickstart) routines and the disk-loadable libraries. These include files are organized on a functional basis. For example, files pertinent to graphics are listed under “graphics/graphicsitem.h.”

This appendix is a hard copy of the “SYS:includes” directory on the Amiga C (Lattice C) disk.

## **Assembly-language Include Files — “.i” Files**

The second portion of this appendix contains the assembly language include files that define the system data structures used by the ROM (or kickstart) routines and the disk-loadable libraries. These include files are organized on a functional basis. For example, files pertinent to graphics are listed under “graphics/graphicsitem.i.”

This appendix is a hard copy of the “SYS:includes” directory on the Amiga Macro Assembler disk.

File numbers for cross-reference listing:

|                 |                 |                |                    |
|-----------------|-----------------|----------------|--------------------|
| 1:adkbits.i     | 2:audio.i       | 3:blit.i       | 4:bootblock.i      |
| 5:cia.i         | 6:ciabase.i     | 7:clip.i       | 8:clipboard.i      |
| 9:console.i     | 10:copper.i     | 11:custom.i    | 12:disk.i          |
| 13:diskfont.i   | 14:display.i    | 15:dmabits.i   | 16:dos.i           |
| 17:dos_lib.i    | 18:dosexten.i   | 19:gameport.i  | 20:gels.i          |
| 21:gfx.i        | 22:gfxbase.i    | 23:icon.i      | 24:input.i         |
| 25:inputevent.i | 26:intbits.i    | 27:intuition.i | 28:intuitionbase.i |
| 29:keyboard.i   | 30:keymap.i     | 31:layers.i    | 32:misc.i          |
| 33:narrator.i   | 34:parallel.i   | 35:potgo.i     | 36:printer.i       |
| 37:prtbase.i    | 38:rastport.i   | 39:regions.i   | 40:serial.i        |
| 41:sprite.i     | 42:startup.i    | 43:text.i      | 44:timer.i         |
| 45:trackdisk.i  | 46:translator.i | 47:view.i      | 48:workbench.i     |

|                        |                                          |
|------------------------|------------------------------------------|
| A,                     | 16-113, 27-217, 27-549, 27-1235, 27-1443 |
| ABC,                   | 3-38                                     |
| aBMS,                  | 36-113                                   |
| ABNC,                  | 3-39                                     |
| ABORT,                 | 34-75, 40-72                             |
| absoluted,             | 27-572, 27-574                           |
| AC,                    | 20-137                                   |
| ac_AnimBob,            | 20-157                                   |
| ac_AnimCRoutine,       | 20-153                                   |
| ac_CompFlags,          | 20-139                                   |
| ac_dat,                | 11-100                                   |
| ac_HeadOb,             | 20-156                                   |
| ac_len,                | 11-97                                    |
| ac_NextComp,           | 20-148                                   |
| ac_NextSeq,            | 20-151                                   |
| ac_per,                | 11-98                                    |
| ac_PrevComp,           | 20-149                                   |
| ac_PrevSeq,            | 20-152                                   |
| ac_ptr,                | 11-96                                    |
| ac_SIZE,               | 20-158                                   |
| ac_SIZEOF,             | 11-101                                   |
| ac_Timer,              | 20-143                                   |
| ac_TimeSet,            | 20-146                                   |
| ac_vol,                | 11-99                                    |
| ac_XTrans,             | 20-155                                   |
| ac_YTrans,             | 20-154                                   |
| aCAM,                  | 36-116                                   |
| ACCESS_READ,           | 16-42                                    |
| ACCESS_WRITE,          | 16-44                                    |
| ACTION_COPY_DIR,       | 18-127                                   |
| ACTION_CREATE_DIR,     | 18-130                                   |
| ACTION_CURRENT_VOLUME, | 18-118                                   |
| ACTION_DELETE_OBJECT,  | 18-124                                   |
| ACTION_DIE,            | 18-116                                   |
| ACTION_DISK_CHANGE,    | 18-141                                   |
| ACTION_DISK_INFO,      | 18-133                                   |
| ACTION_DISK_TYPE,      | 18-140                                   |
| ACTION_EVENT,          | 18-117                                   |
| ACTION_EXAMINE_NEXT,   | 18-132                                   |

|                        |                     |
|------------------------|---------------------|
| ACTION_EXAMINE_OBJECT, | 18-131              |
| ACTION_FREE_LOCK,      | 18-123              |
| ACTION_GET_BLOCK,      | 18-114              |
| ACTION_INFO,           | 18-134              |
| ACTION_INHIBIT,        | 18-139              |
| ACTION_LOCATE_OBJECT,  | 18-119              |
| ACTION_NIL,            | 18-113              |
| ACTION_PARENT,         | 18-137              |
| ACTION_READ,           | 18-122              |
| ACTION_RENAME_DISK,    | 18-120              |
| ACTION_RENAME_OBJECT,  | 18-125              |
| ACTION_SET_COMMENT,    | 18-136              |
| ACTION_SET_MAP,        | 18-115              |
| ACTION_SET_PROTECT,    | 18-129              |
| ACTION_TIMER,          | 18-138              |
| ACTION_WAIT_CHAR,      | 18-128              |
| ACTION_WRITE,          | 18-121              |
| ACTIVATE,              | 27-1148             |
| ACTIVE,                | 12-72, 34-76, 40-73 |
| ACTIVIEWINDOW,         | 27-932              |
| actually,              | 27-1488             |
| ADALLOC_MAXPREC,       | 2-19                |
| ADALLOC_MINPREC,       | 2-18                |
| ADCMD_ALLOCATE,        | 2-29                |
| ADCMD_FINISH,          | 2-23                |
| ADCMD_FREE,            | 2-21                |
| ADCMD_LOCK,            | 2-25                |
| ADCMD_PERVOL,          | 2-24                |
| ADCMD_SETPREC,         | 2-22                |
| ADCMD_WAITCYCLE,       | 2-26                |
| ADCMDB_NOUNIT,         | 2-27                |
| ADCMDF_NOUNIT,         | 2-28, 2-29          |
| added,                 | 27-515              |
| address,               | 27-1237             |
| aDEN1,                 | 36-71               |
| aDEN2,                 | 36-70               |
| aDEN3,                 | 36-69               |
| aDEN4,                 | 36-68               |
| aDENS,                 | 36-66               |
| ADHARD_CHANNELS,       | 2-16                |
| ADIOB_NOWAIT,          | 2-35                |
| ADIOB_PERVOL,          | 2-31                |
| ADIOB_SYNCYCLE,        | 2-33                |
| ADIOB_WRITEMESSAGE,    | 2-37                |
| ADIOERR_ALLOCFAILED,   | 2-41                |
| ADIOERR_CHANNELSTOLEN, | 2-42                |
| ADIOERR_NOALLOCATION,  | 2-40                |
| ADIOF_NOWAIT,          | 2-36                |
| ADIOF_PERVOL,          | 2-32                |
| ADIOF_SYNCYCLE,        | 2-34                |
| ADIOF_WRITEMESSAGE,    | 2-38                |
| ADKB_FAST,             | 1-22                |
| ADKB_MFMPREC,          | 1-18                |
| ADKB_MSBSYNC,          | 1-21                |
| ADKB_PRECOMP,          | 1-17                |

```

ADKB_PRECOMP1, 1-16
ADKB_SETCLR, 1-15
ADKB_UARTBRK, 1-19
ADKB_USE0P1, 1-26
ADKB_USE0V1, 1-30
ADKB_USE1P2, 1-25
ADKB_USE1V2, 1-29
ADKB_USE2P3, 1-24
ADKB_USE2V3, 1-28
ADKB_USE3PN, 1-23
ADKB_USE3VN, 1-27
ADKB_WORDSYNC, 1-20
 adkcon, 11-87
 adkconr, 11-28
 ADKF_FAST, 1-39
 ADKF_MEMPREC, 1-35
 ADKF_MSBSYNC, 1-38
ADKF_PRE000NS, 1-49
ADKF_PRE140NS, 1-50
ADKF_PRE280NS, 1-51
ADKF_PRE560NS, 1-52
ADKF_PRECOMP0, 1-34, 1-50, 1-52
ADKF_PRECOMP1, 1-33, 1-51, 1-52
 ADKF_SETCLR, 1-32
 ADKF_UARTBRK, 1-36
 ADKF_USE0P1, 1-43
 ADKF_USE0V1, 1-47
 ADKF_USE1P2, 1-42
 ADKF_USE1V2, 1-46
 ADKF_USE2P3, 1-41
 ADKF_USE2V3, 1-45
 ADKF_USE3PN, 1-40
 ADKF_USE3VN, 1-44
ADKF_WORDSYNC, 1-37
 advance, 27-559
 aEXTEND, 36-126
 AF, 13-60, 13-61, 13-63
 af_Attr, 13-65
 af_SIZEOF, 13-66
 af_Type, 13-64
 AFH, 13-68
 afh_AF, 13-70
afh_NumEntries, 13-69
 aENT0, 36-81
 aENT1, 36-82
 aENT10, 36-91
 aENT2, 36-83
 aENT3, 36-84
 aENT4, 36-85
 aENT5, 36-86
 aENT6, 36-87
 aENT7, 36-88
 aENT8, 36-89
 aENT9, 36-90
 AGNUS, 21-12
 aHTS, 36-118

```

```

 ai_Count, 38-98
 ai_FirstX, 38-100
 ai_FirstY, 38-101
 ai_FlagPtr, 38-97
 ai_FlagTbl, 38-96
 ai_MaxCount, 38-99
 ai_SIZEOF, 38-102
 ai_VctrPtr, 38-95
 ai_VctrTbl, 38-94
 aIND, 36-44
 aJFY0, 36-100
 aJFY2, 36-101
 aJFY3, 36-102
 aJFY5, 36-97
 aJFY6, 36-99
 aJFY7, 36-98
 ALERT_TYPE, 27-1720
 algorithmic, 27-312
 all, 27-794
 ALLOC0, 12-68
 ALLOC1, 12-69
 ALLOC2, 12-70
 ALLOC3, 12-71
 allocated, 27-1655
 ALLOWED, 27-470
 aLMS, 36-110
 ALPHA_P_101, 27-1630
 ALTKEYMAP, 27-463
 ANBC, 3-40
 ANBNC, 3-41
 ANEL, 36-45
 ANFRACSIZE, 20-41
 ANIMHALE, 20-42
 any, 27-723
 AO, 20-162
ao_AnimORoutine, 20-180
 ao_AnOldX, 20-169
 ao_AnOldY, 20-168
 ao_AnX, 20-172
 ao_AnY, 20-171
 ao_AUserExt, 20-182
 ao_Clock, 20-167
 ao_HeadComp, 20-181
 ao_NextOb, 20-164
 ao_PrevOb, 20-165
 ao_RingXTrans, 20-179
 ao_RingYTrans, 20-178
 ao_SIZEOF, 20-183
 ao_XAccel, 20-176
 ao_XVel, 20-175
 ao_YAccel, 20-177
 ao_YVel, 20-174
 aPERF, 36-107
 aPERF0, 36-108
 aPLD, 36-79
 aPLU, 36-78

```

```

appear, 27-797
aPROP0, 36-95
aPROP1, 36-94
aPROP2, 36-93
 are, 27-873, 28-58
 area, 27-563
AreaInfo, 38-93
AREAOUTLINE, 38-46
 aren, 27-132, 27-229, 27-682, 27-1336
 aRI, 36-46
 aRIN, 36-43
 aRIS, 36-42
 aRMS, 36-111
 as, 27-305, 27-305, 27-774, 28-28
 ASAP, 27-966
 aSBC, 36-56
 aSFC, 36-55
 aSGR0, 36-48
 aSGR1, 36-53
 aSGR2, 36-54
 aSGR23, 36-50
 aSGR24, 36-52
 aSGR3, 36-49
 aSGR4, 36-51
ASHIFTSHIFT, 3-63
 aSHORP0, 36-58
 aSHORP1, 36-60
 aSHORP2, 36-59
 aSHORP3, 36-62
 aSHORP4, 36-61
 aSHORP5, 36-64
 aSHORP6, 36-63
 aSLPF, 36-106
 aSLRM, 36-115
ASPECT_HORIZ, 27-1613
ASPECT_VERT, 27-1614
 associated, 27-1027
 aSTBM, 36-114
 aSUS0, 36-77
 aSUS1, 36-74
 aSUS2, 36-73
 aSUS3, 36-76
 aSUS4, 36-75
 aTBC0, 36-120
 aTBC1, 36-122
 aTBC3, 36-121
 aTBC4, 36-123
 aTBCALL, 36-124
 aTBSALL, 36-125
 aTMS, 36-112
 aTSS, 36-96
 aud, 11-99
 aud0, 11-90
 aud1, 11-91
 aud2, 11-92
 aud3, 11-93

```

```

audio, 2-13
AUDIONAME, 2-12
 AUL, 3-78
AUTOBACKPEN, 27-1735
AUTODRAWMODE, 27-1736
AUTOFRONTPEN, 27-1734
AUTOITEXTFONT, 27-1739
 AUTOKNOB, 27-541, 27-582
 AUTOLEFTEDGE, 27-1737
 automatic, 27-134, 27-231, 27-684, 27-1338
 AUTONEXTTEXT, 27-1740
 AUTOTOPEDGE, 27-1738
 aVERP0, 36-104
 aVERP1, 36-105
 aVTS, 36-119
b_BobFlags, 20-55
BACKDROP, 27-1137
Background, 48-131
BACKSAVED, 20-19
 baggage, 27-1355
BAUD_110, 27-1578
BAUD_1200, 27-1580
BAUD_19200, 27-1584
BAUD_2400, 27-1581
BAUD_300, 27-1579
BAUD_4800, 27-1582
BAUD_9600, 27-1583
BAUD_MIDI, 27-1585
BB, 4-29
BB_CHKSUM, 4-31
BB_DOSBLOCK, 4-32
BB_ENTRY, 4-33
BB_ID, 4-30
BB_SIZE, 4-34
BBID_DOS, 4-38
BBID_KICK, 4-42
BBNAME_DOS, 4-47
BBNAME_KICK, 4-48
BCOB_DEST, 3-47
BCOB_SRC, 3-50
BCOB_SRCB, 3-49
BCOB_SRCC, 3-48
BCOF_DEST, 3-51
BCOF_SRC, 3-54
BCOF_SRCB, 3-53
BCOF_SRCC, 3-52
BCIF_DESC, 3-56
bd_BackPen, 27-737
bd_Count, 27-739
bd_DrawMode, 27-738
bd_FrontPen, 27-736
bd_LeftEdge, 27-734
bd_NextBorder, 27-742
bd_SIZEOF, 27-749
bd_TopEdge, 27-735
bd_XY, 27-740

```

BDRAWN, 20-32  
 been, 27-1655  
 BEEPING, 27-1401  
 before, 27-224  
 below, 27-512  
 between, 27-253  
 BF\_BOBSAWAY, 20-55  
 bit, 27-329, 27-805, 27-809  
 BITCLR, 21-11  
 BitMap, 21-15, 27-1021, 27-1244, 27-1404, 27-1443, 27-1447  
 bits, 27-170, 27-325, 27-771  
 BITSET, 21-10  
 BITSPERBYTE, 16-34  
 BITSPERLONG, 16-36  
 BLITREVERSE, 3-74  
 bitadat, 11-70  
 bitafwm, 11-55  
 bitalwm, 11-56  
 bitamod, 11-65  
 bitapt, 11-59  
 bitbdat, 11-69  
 bitbmod, 11-64  
 bitbpt, 11-58  
 bitcdat, 11-68  
 bitcmmod, 11-63  
 bitcon0, 11-53  
 bitcon1, 11-54  
 bitcpt, 11-57  
 bitddat, 11-19  
 bitdmod, 11-66  
 bitdpt, 11-60  
 bitnode, 3-14  
 bitsize, 11-61  
 bm\_BytesPerRow, 21-16  
 bm\_Depth, 21-19  
 bm\_Flags, 21-18  
 bm\_Pad, 21-20  
 bm\_Planes, 21-21  
 bm\_Rows, 21-17  
 bm\_SIZEOF, 21-22, 27-1354  
 bn\_beamsync, 3-20  
 bn\_bitsize, 3-19  
 bn\_cleanup, 3-21  
 bn\_dummy, 3-18  
 bn\_function, 3-16  
 bn\_n, 3-15  
 bn\_SIZEOF, 3-22  
 bn\_stat, 3-17  
 BOB, 20-113  
 bob\_After, 20-125  
 bob\_Before, 20-123  
 bob\_BobComp, 20-128  
 bob\_BobFlags, 20-116  
 bob\_BobVSprite, 20-127  
 bob\_BUserExt, 20-132  
 bob\_DBuffer, 20-130

bob\_ImageShadow, 20-120  
 bob\_SaveBuffer, 20-118  
 bob\_SavePlanes, 20-115  
 bob\_SIZEOF, 20-133  
 BOBISCOMP, 20-29  
 BOBNIX, 20-34  
 BOBSAWAY, 20-33  
 BOBUPDATE, 20-20  
 BOLD, 43-21  
 BOOL, 27-1539  
 BOOLGADGET, 27-492  
 BOOTSECTS, 4-36  
 Border, 27-305, 27-448, 27-732, 27-1146  
 BORDERLESS, 27-1145  
 BorderTop, 27-1029  
 BOTTOMBORDER, 27-453  
 box, 27-173  
 bpl1mod, 11-108  
 bpl2mod, 11-109  
 bplcon0, 11-105  
 bplcon1, 11-106  
 bplcon2, 11-107  
 bpldat, 11-111  
 bplpt, 11-103  
 broadcast, 27-280  
 BROTHER\_15XL, 27-1631  
 BSHIFTSHIFT, 3-64  
 BSTR, 16-84, 18-183, 18-186, 18-188, 18-191, 18-216  
 Buffer, 27-627, 27-1475  
 BUSERFLACS, 20-27  
 BWAITING, 20-31  
 by, 27-251, 27-349, 27-395, 27-940, 27-1029  
 BYTESPERLONG, 16-35  
 C, 4-48, 16-116, 27-19  
 call, 27-730, 28-32  
 called, 28-37  
 CBD\_CURRENTREADID, 8-29  
 CBD\_CURRENTWRITEID, 8-30  
 CBD\_POST, 8-28  
 CBERR\_OBSOLETEID, 8-32  
 CBM\_MPS1000, 27-1632  
 CD\_ASKKEYMAP, 9-26  
 CD\_SETKEYMAP, 9-27  
 changing, 27-1490  
 character, 27-621  
 check, 27-947  
 CHECKED, 27-179  
 CHECKIT, 27-155, 27-1701  
 CheckMark, 27-1703  
 checkmarked, 27-1077, 27-1224  
 CHECKWIDTH, 27-1709  
 ci\_DestAddr, 10-19  
 ci\_DestData, 10-22  
 ci\_HWaitPos, 10-21  
 ci\_nxtlist, 10-17  
 ci\_OpCode, 10-16

```

ci_SIZEOF, 10-24
ci_VWaitPos, 10-18
 ciaa, 5-7
 CIAA_NAME, 5-6
 ciab, 5-11
 CIAB_NAME, 5-10
 CIAR, 6-18
 cl_, 10-36
cl_CopList, 10-35
cl_CopIns, 10-37
cl_CopLStart, 10-39
cl_CopPtr, 10-38
cl_CopSStart, 10-40
cl_Count, 10-41
cl_DyOffset, 10-43
cl_MaxCount, 10-42
cl_Next, 10-34
cl_SIZEOF, 10-44
 CLEANME, 3-26
 CLEANMEN, 3-25, 3-26
 cleared, 27-943
cli_Background, 18-193
cli_CommandDir, 18-184
cli_CommandFile, 18-191
cli_CommandName, 18-186
cli_CurrentInput, 18-190
cli_CurrentOutput, 18-194
cli_DefaultStack, 18-195
cli_FailLevel, 18-187
cli_Interactive, 18-192
cli_Module, 18-197
cli_Prompt, 18-188
cli_Result2, 18-182
cli_ReturnCode, 18-185
cli_SetName, 18-183
cli_SIZEOF, 18-198
cli_StandardInput, 18-189
cli_StandardOutput, 18-196
 clicks, 27-1515
 clip, 27-29
ClipboardUnitPartial, 8-35
 ClipRect, 7-50
 Close, 17-19, 27-490
CLOSEWINDOW, 27-917
 CLR, 20-50
 clxcon, 11-84
 clxdat, 11-26
cm_ColorTable, 47-24
 cm_Count, 47-23
 cm_Flags, 47-21
 cm_SIZEOF, 47-25
 cm_Type, 47-22
 CMD_CLEAR, 45-98
 CMD_READ, 45-93
 CMD_UPDATE, 45-97
 CMD_WRITE, 45-92

```

```

collTable, 47-55
 color, 11-122, 37-110
 ColorMap, 47-20
 COLORON, 14-21
 colors, 27-786
CommandLineInterface, 18-181
 COMMSEQ, 27-163
 COMMWIDTH, 27-1710
conjunction, 27-411
Container, 27-629
containing, 27-327
coordinates, 27-1089
 cop1lc, 11-74
 cop2lc, 11-75
 copcon, 11-43
 copinit, 10-53
 copinit_diagstr, 10-54
 copinit_SIZEOF, 10-57
 copinit_sprstop, 10-56
 copinit_sprstrtop, 10-55
 CopIns, 10-15, 11-78
 copjmp1, 11-76
 copjmp2, 11-77
 CopList, 10-33
COPPER_MOVE, 10-9
COPPER_WAIT, 10-10
 correct, 27-1021
 count, 17-10, 17-12, 17-13
count-vsizer, 17-13
cp_collPtrs, 47-56
cp_SIZEOF, 47-57
CPR_NT_LOE, 10-12
CPR_NT_SHT, 10-13
 cprlist, 10-27
CPRNXTBUE, 10-11
 cr_p1, 7-59
 cr_p2, 7-60
 cr_BitMap, 7-54
 cr_Flags, 7-62
 CR_HWADDR, 6-19
 CR_IActive, 6-22
 CR_IEnable, 6-21
 CR_IntMask, 6-20
 CR_INTNODE, 6-23
 CR_IVALRM, 6-26
 CR_IVFLG, 6-28
 CR_IVSP, 6-27
 CR_IVTA, 6-24
 CR_IVTB, 6-25
 cr_LObs, 7-53
 cr_MaxX, 7-57
 cr_MaxY, 7-58
 cr_MinX, 7-55
 cr_MinY, 7-56
 cr_Next, 7-51
 cr_Prev, 7-52

```



```

cr_reserved, 7-61
CR_SIZE, 6-29
cr_SIZEOF, 7-63
CreateDir, 17-33
CreateProc, 17-36
 crl_max, 10-30
 crl_Next, 10-28
 crl_SIZEOF, 10-31
 crl_start, 10-29
CTC_HCLRTAB, 9-86
CTC_HCLRTABSALL, 9-87
CTC_HSETTAB, 9-85
 CTRL_C, 16-150
 CTRL_D, 16-151
 CTRL_E, 16-152
 CTRL_F, 16-153
cu_Node, 8-36
CURRENT, 37-63
CurrentDir, 17-34
CUSTOM, 27-1626
CUSTOM_NAME, 27-1629
CUSTOMBITMAP, 27-1403, 27-1445
CUSTOMSCREEN, 27-1235, 27-1396
 D, 4-47, 16-113, 16-114, 16-115
 data, 27-778, 27-790, 27-819, 27-1379, 28-28
 DateStamp, 16-47, 17-45
 DBLPE, 14-22
 DBP, 20-190
dbp_BufBuffer, 20-197
dbp_BufPath, 20-193
dbp_BufPlanes, 20-199
 dbp_BufX, 20-192
 dbp_BufY, 20-191
 dbp_SIZEOF, 20-200
 DBUFFER, 38-44
dd_Children, 48-66
dd_CmdBytes, 37-51
dd_CmdVectors, 37-50
dd_CurrentX, 48-48
dd_CurrentY, 48-49
dd_DownMove, 48-57
dd_DrawerWin, 48-64
dd_ExecBase, 37-49
dd_HorizImage, 48-60
dd_HorizProp, 48-62
dd_HorizScroll, 48-54
dd_LeftMove, 48-58
 dd_Lock, 48-67
 dd_MaxX, 48-52
 dd_MaxY, 48-53
 dd_MinX, 48-50
 dd_MinY, 48-51
dd_NewWindow, 48-47
dd_NumCommands, 37-52
dd_Object, 48-65
dd_RightMove, 48-59

```

```

dd_Segment, 37-48
dd_SIZEOF, 37-53, 37-80, 48-68
dd_UpMove, 48-56
dd_VertImage, 48-61
dd_VertProp, 48-63
dd_VertScroll, 48-55
 ddfstop, 11-82
 ddfstrt, 11-81
DEADEND_ALERT, 27-1723
 decide, 27-545
DEFERREFRESH, 27-279
 DEFREQ, 33-17
 define, 27-249
 DEFMODE, 33-23
 DEFPTCH, 33-14
 DEFERATE, 33-15
 DEFSEX, 33-22
 DEFVOL, 33-16
 Delay, 17-46
 DELETE, 16-74
DeleteFile, 17-25
DELTAMOVE, 27-936
DENISE, 21-13
describe, 27-323, 27-1478
DESIGNED, 43-31
 DEST, 3-58
 device, 2-13, 34-68, 40-60, 44-27, 45-70
DeviceData, 37-47
DeviceProc, 17-42
 devices, 25-14, 27-57, 27-61, 37-31, 37-34, 37-37
DEVINIT, 8-26, 9-24, 19-19, 24-17, 29-17, 36-35, 44-41, 45-76
DevList, 18-207
DevList_SIZEOF, 18-217
 dfh_DF, 13-51
 dfh_FileID, 13-52
 DFH_ID, 13-40
 dfh_Name, 13-55
dfh_Revision, 13-53
dfh_Segment, 13-54
dfh_SIZEOF, 13-57
 dfh_TF, 13-56
DETCH_MASK, 14-37
di_Devices, 18-173
di_DeVInfo, 18-172
di_Handlers, 18-174
 di_McName, 18-171
 di_NetHand, 18-175
 di_SIZEOF, 18-176
DIAB_630, 27-1633
DIAB_ADV_D25, 27-1634
DIAB_C_150, 27-1635
DISCRESOURCE, 12-55
DISCRESOURCEUNIT, 12-47
 disk, 12-101, 13-61
DISKFONT, 43-26
DiskFontHeader, 13-43

```

DISKINSERTED, 27-926  
 DISKNAME, 12-100  
 DiskObject, 48-74  
 DISKREMOVED, 27-928  
 display, 27-788, 27-805, 27-1352  
 DisplayAlert, 27-1716  
 displayed, 27-122  
 DIW\_HORIZ\_POS, 14-32  
 DIW\_VRTCL\_POS, 14-33  
 DIW\_VRTCL\_POS\_SHIFT, 14-34  
 divstop, 11-80  
 divstrt, 11-79  
 dl\_A2, 18-153  
 dl\_A5, 18-154  
 dl\_A6, 18-155  
 dl\_DiskType, 18-214  
 dl\_GV, 18-152  
 dl\_lib, 18-150  
 dl\_Lock, 18-211  
 dl\_LockList, 18-213  
 dl\_Name, 18-216  
 dl\_Next, 18-208  
 dl\_Root, 18-151  
 dl\_SIZEOF, 18-156  
 dl\_Task, 18-210  
 dl\_Type, 18-209  
 dl\_unused, 18-215  
 dl\_VolumeDate, 18-212  
 DLT\_DEVICE, 18-220  
 DLT\_DIRECTORY, 18-221  
 DLT\_VOLUME, 18-222  
 DMAB\_AUDIO, 15-39  
 DMAB\_AUD1, 15-40  
 DMAB\_AUD2, 15-41  
 DMAB\_AUD3, 15-42  
 DMAB\_BLITHOG, 15-49  
 DMAB\_BLITTER, 15-45  
 DMAB\_BLTDONE, 15-50  
 DMAB\_BLTNZERO, 15-51  
 DMAB\_COPPER, 15-46  
 DMAB\_DISK, 15-43  
 DMAB\_MASTER, 15-48  
 DMAB\_RASTER, 15-47  
 DMAB\_SETCLR, 15-38  
 DMAB\_SPRITE, 15-44  
 dmacon, 11-83  
 dmaconr, 11-20  
 DMAF\_ALL, 15-31  
 DMAF\_AUDIO, 15-20  
 DMAF\_AUD1, 15-21  
 DMAF\_AUD2, 15-22  
 DMAF\_AUD3, 15-23  
 DMAF\_AUDIO, 15-19  
 DMAF\_BLITHOG, 15-30  
 DMAF\_BLITTER, 15-26  
 DMAF\_BLTDONE, 15-35

Apr 29 10:21 1986 i.xref Page 14

DMAF\_BLTNZERO, 15-36  
 DMAF\_COPPER, 15-27  
 DMAF\_DISK, 15-24  
 DMAF\_MASTER, 15-29  
 DMAF\_RASTER, 15-28  
 DMAF\_SETCLR, 15-18  
 DMAF\_SPRITE, 15-25  
 do\_CurrentX, 48-81  
 do\_CurrentY, 48-82  
 do\_DefaultTool, 48-79  
 do\_DrawerData, 48-83  
 do\_Cadget, 48-77  
 do\_Magic, 48-75  
 do\_SIZEOF, 48-86  
 do\_StackSize, 48-85  
 do\_ToolTypes, 48-80  
 do\_ToolWindow, 48-84  
 do\_Type, 48-78  
 do\_Version, 48-76  
 does, 27-143, 27-240, 27-697, 27-727, 27-1347  
 doesn, 27-1246  
 don, 27-440, 27-653  
 DONE, 37-65  
 DOS, 4-39, 16-16, 18-25, 42-23  
 dosextens, 37-40  
 DosInfo, 18-170  
 DosLibrary, 18-149  
 DOSNAME, 16-15  
 DosPacket, 18-76  
 DOWNKEYS, 19-30  
 dp\_Action, 18-90  
 dp\_Arg1, 18-88, 18-93  
 dp\_Arg2, 18-94  
 dp\_Arg3, 18-95  
 dp\_Arg4, 18-96  
 dp\_Arg5, 18-97  
 dp\_Arg6, 18-98  
 dp\_Arg7, 18-99  
 dp\_BufAddr, 18-93  
 dp\_Link, 18-77  
 dp\_Port, 18-78  
 dp\_Res1, 18-82, 18-91  
 dp\_Res2, 18-86, 18-92  
 dp\_SIZEOF, 18-100, 18-108  
 dp\_Status, 18-91  
 dp\_Status2, 18-92  
 dp\_Type, 18-80, 18-90  
 DR, 12-68, 12-69, 12-70, 12-71, 12-72  
 DR\_ALLOCUNIT, 12-106  
 DR\_CIARESOURCE, 12-60  
 DR\_CURRENT, 12-56  
 DR\_DISCLOCK, 12-63  
 DR\_DISCSYNC, 12-64  
 DR\_FLAGS, 12-57  
 DR\_FREEUNIT, 12-107  
 DR\_GETUNIT, 12-108

DR\_GETUNITID, 12-110  
 DR\_GIVEUNIT, 12-109, 12-112  
 DR\_INDEX, 12-65  
 DR\_LASTCOMM, 12-112  
 DR\_pad, 12-58  
 DR\_SIZE, 12-66  
 DR\_SYSLIB, 12-59  
 DR\_UNITID, 12-61  
 DR\_WAITING, 12-62  
 DRAFT, 27-1601  
 DrawBorder, 27-730  
 DrawerData, 48-46  
 DRAWERDATAFILESIZE, 48-71  
 DrawerOpen, 48-129  
 drawn, 27-184  
 draws, 27-224, 27-727  
 DRT\_37422D2S, 12-122  
 DRT\_AMIGA, 12-121  
 DRT\_EMPTY, 12-123  
 DRU\_DISCLOCK, 12-48  
 DRU\_DISCSYNC, 12-49  
 DRU\_INDEX, 12-50  
 DRU\_SIZE, 12-51  
 DS, 12-102, 35-10, 44-28, 45-71  
 ds\_Days, 16-48  
 ds\_Minute, 16-49  
 ds\_SIZEOF, 16-51, 16-64, 18-165, 18-212  
 ds\_Tick, 16-50  
 dskbytr, 11-33  
 dskdat, 11-39  
 dskdatr, 11-23  
 DSKDMAOFF, 12-82  
 dsklen, 11-38  
 dskpt, 11-37  
 dsksync, 11-72  
 DSR\_CPR, 9-82  
 DU, 37-68  
 du\_Flags, 37-59  
 duplicate, 27-136, 27-233, 27-686, 27-1261, 27-1340  
 DupLock, 17-29  
 each, 27-805  
 EIGHT\_LPI, 27-1606  
 either, 27-305, 27-1482  
 ELITE, 27-1593  
 EnableCLI, 27-1539  
 ENDGADGET, 27-421  
 entry, 27-616  
 EOFMODE, 34-73, 40-64  
 EPSON, 27-1636  
 EPSON\_JX\_80, 27-1637  
 ERROR\_ACTION\_NOT\_KNOWN, 16-123  
 ERROR\_COMMENT\_TOO\_BIG, 16-133  
 ERROR\_DELETE\_PROTECTED, 16-135  
 ERROR\_DEVICE\_NOT\_MOUNTED, 16-131  
 ERROR\_DIRECTORY\_NOT\_EMPTY, 16-130  
 ERROR\_DISK\_FULL, 16-134

ERROR\_DISK\_NOT\_VALIDATED, 16-127  
 ERROR\_DISK\_WRITE\_PROTECTED, 16-128  
 ERROR\_INVALID\_COMPONENT\_NAME, 16-124  
 ERROR\_INVALID\_LOCK, 16-125  
 ERROR\_NO\_DISK, 16-139  
 ERROR\_NO\_FREE\_STORE, 16-119  
 ERROR\_NO\_MORE\_ENTRIES, 16-140  
 ERROR\_NOT\_A\_DOS\_DISK, 16-138  
 ERROR\_OBJECT\_EXISTS, 16-121  
 ERROR\_OBJECT\_IN\_USE, 16-120  
 ERROR\_OBJECT\_NOT\_FOUND, 16-122  
 ERROR\_OBJECT\_WRONG\_TYPE, 16-126  
 ERROR\_READ\_PROTECTED, 16-137  
 ERROR\_RENAME\_ACROSS\_DEVICES, 16-129  
 ERROR\_SEEK\_ERROR, 16-132  
 ERROR\_WRITE\_PROTECTED, 16-136  
 ETD\_CLEAR, 45-98  
 ETD\_FORMAT, 45-96  
 ETD\_MOTOR, 45-94  
 ETD\_READ, 45-93  
 ETD\_SEEK, 45-95  
 ETD\_UPDATE, 45-97  
 ETD\_WRITE, 45-92  
 event, 27-442  
 Examine, 17-30  
 examined, 27-807  
 except, 27-849  
 EXCLUSIVE\_LOCK, 16-43  
 EXEC\_LIBRARIES\_1, 12-36, 18-20, 22-12, 28-19, 32-9, 37-23  
 EXECUTE, 16-73, 17-50  
 Exit, 17-37, 27-1657  
 ExNext, 17-31  
 EXPUNGED, 37-78  
 EXTCOM, 45-74  
 EXTENDED, 43-19  
 FANFOLD, 27-1588  
 FC, 13-25  
 fc\_FileName, 13-26  
 fc\_Flags, 13-29  
 fc\_SIZEOF, 13-30  
 fc\_Style, 13-28  
 fc\_YSize, 13-27  
 FCH, 13-34  
 fch\_FC, 13-37  
 fch\_FileID, 13-35  
 FCH\_ID, 13-32  
 fch\_NumEntries, 13-36  
 FEMALE, 33-21  
 fh\_Arg1, 18-71  
 fh\_Arg2, 18-72  
 fh\_Args, 18-70, 18-71  
 fh\_Buf, 18-63  
 fh\_End, 18-65  
 fh\_Func1, 18-67  
 fh\_Func2, 18-68  
 fh\_Func3, 18-69

```

 fh_Funcs, 18-66, 18-67
 fh_Interactive, 18-61
 fh_Link, 18-60
 fh_Pos, 18-64
 fh_SIZEOF, 18-73
 fh_Type, 18-62
 FIB, 16-71, 16-72, 16-73, 16-74
 fib_Comment, 16-65
 fib_DateStamp, 16-64
 fib_DirEntryType, 16-57
 fib_DiskKey, 16-56
 fib_EntryType, 16-61
 fib_FileName, 16-59
 fib_NumBlocks, 16-63
 fib_Protection, 16-60
 fib_Size, 16-62
 fib_SIZEOF, 16-67
 FileHandle, 18-59
 FileInfoBlock, 16-55
 FileLock, 18-226
 FILENAME_SIZE, 27-1472, 27-1543
 files, 27-140, 27-237, 27-690, 27-1344
 FILL_CARRYIN, 3-70
 FILL_OR, 3-68
 FILL_XOR, 3-69
 filled, 27-817
 final, 27-614
 FINE, 27-1594
 fl_Access, 18-229
 fl_Key, 18-228
 fl_Link, 18-227
 fl_MemList, 48-93
 fl_NumFree, 48-92
 fl_SIZEOF, 18-232
 fl_Task, 18-230
 fl_Volume, 18-231
 flag, 27-127
 Flags, 27-849, 27-1044, 27-1217
 FOLLOWMOUSE, 27-436, 27-443
 FontSize, 27-1486
 four, 27-792
 FP, 43-25, 43-26, 43-27, 43-28, 43-29, 43-30,
 43-31, 43-32
 FREEHORIZ, 27-583
 FreeList, 48-91
 FreeList_SIZEOF, 48-95, 48-121
 FREEVERT, 27-588
 from, 27-1722
 FRST_DOT, 38-42, 38-107
 FRST_DOTn, 38-106
 FS, 43-19, 43-20, 43-21, 43-22
 FS_NORMAL, 43-18
 function, 27-430
 functions, 27-866
 GADGBACKFILL, 48-163
 GADGDISABLED, 27-399

```

```

 Gadget, 27-289, 27-333, 27-368, 27-409, 27-416,
 27-506, 27-547, 27-565, 27-643, 27-646,
 27-651, 27-1253
 GADGET0002, 27-493
 GADGETDOWN, 27-911
 gadgets, 27-217, 27-814, 27-1728
 GadgetType, 27-468, 27-473
 GADGETUP, 27-912
 GADGHBOX, 27-358
 GADGHCOMP, 27-357
 GADGHIGHBITS, 27-356
 GADGHIMAGE, 27-359
 GADGHNONE, 27-360
 GADGIMAGE, 27-365
 GADGIMMEDIATE, 27-413
 GamePortTrigger, 19-33
 gb_ActiView, 22-20
 gb_BeamSync, 22-38
 gb_BlitLock, 22-44
 gb_BlitNest, 22-45
 gb_BlitOwner, 22-47
 gb_blitter, 22-23
 gb_BlitWaitQ, 22-46
 gb_bltnd, 22-26
 gb_bltshr, 22-32
 gb_blttl, 22-27
 gb_bsblthd, 22-28
 gb_bsblttl, 22-29
 gb_bytereserved, 22-41
 gb_cia, 22-22
 gb_copinit, 22-21
 gb_Debug, 22-37
 gb_DefaultFont, 22-34
 gb_DisplayFlags, 22-49
 gb_Flags, 22-43
 gb_LOFlist, 22-24
 gb_Modes, 22-35
 gb_reserved, 22-51
 gb_SHElist, 22-25
 gb_SIZE, 22-52
 gb_SpriteReserved, 22-40
 gb_system_bplcon0, 22-39
 gb_TextFonts, 22-33
 gb_timsrv, 22-31
 gb_TOF_WaitQ, 22-48
 gb_VBlank, 22-36
 gb_vbsrv, 22-30
 GELGONE, 20-21
 GelsInfo, 38-22
 GENLOCK_VIDEO, 47-18
 get, 27-871, 27-1730
 GetPacket, 17-40
 gfx, 7-9, 27-25, 37-109, 38-10, 39-10, 47-9
 GfxBase, 22-19
 gg_Activation, 27-300
 gg_Flags, 27-298

```

```

gg_GadgetID, 27-347
gg_GadgetRender, 27-309
gg_GadgetText, 27-316
gg_GadgetType, 27-302
gg_Height, 27-296
gg_LeftEdge, 27-293
gg_MutualExclude, 27-339
gg_NextGadget, 27-291
gg_SelectRender, 27-314
gg_SIZEOF, 27-351, 48-54, 48-55, 48-56, 48-57, 48-58,
48-59, 48-77, 48-120
gg_SpecialInfo, 27-345
gg_TopEdge, 27-294
gg_UserData, 27-348
gg_Width, 27-295
gi_bottommost, 38-36
gi_collHandler, 38-32
gi_firstBlissObj, 38-37
gi_Flags, 38-25
gi_gelHead, 38-26
gi_gelTail, 38-27
gi_lastBlissObj, 38-38
gi_lastColor, 38-31
gi_leftmost, 38-33
gi_nextLine, 38-29
gi_rightmost, 38-34
gi_SIZEOF, 38-39
gi_sprRsrvd, 38-23
gi_topmost, 38-35
GID_DOWNSCROLL, 48-152
GID_HORIZSCROLL, 48-147
GID_LEFTSCROLL, 48-149
GID_NAME, 48-153
GID_RIGHTSCROLL, 48-150
GID_UPSCROLL, 48-151
GID_VERTSCROLL, 48-148
GID_WBOBJECT, 48-146
GIMMEZEROZERO, 27-1023, 27-1143
GPCT_ABSJOYSTICK, 19-46
GPCT_ALLOCATED, 19-41
GPCT_MOUSE, 19-44
GPCT_NOCONTROLLER, 19-42
GPCT_RELJOYSTICK, 19-45
GPD_ASKCTYPE, 19-22
GPD_ASKTRIGGER, 19-24
GPD_READEVENT, 19-21
GPD_SETCTYPE, 19-23
GPD_SETTRIGGER, 19-25
GPDERR_SETCTYPE, 19-50
GPT, 19-30, 19-31
gpt_Keys, 19-34
gpt_SIZEOF, 19-38
gpt_Timeout, 19-35
gpt_XDelta, 19-36
gpt_YDelta, 19-37
graphics, 7-9, 13-20, 27-25, 27-29, 27-33, 27-37,

```

```

27-41, 27-45, 28-24, 38-10, 39-10, 47-9
GRELBOTTOM, 27-377
GRELHEIGHT, 27-386
GRELRIGHT, 27-379
GRELWIDTH, 27-383
Guide, 27-902, 27-904, 27-927, 27-931, 27-933, 27-935
GZZGADGET, 27-476
have, 27-432, 27-790, 27-871, 27-1246, 27-1267
here, 27-947
HIGHBOX, 27-174
HIGHCOMP, 27-172
HIGHLAGS, 27-169
HIGHIMAGE, 27-171
HIGHLIGHT, 27-185
HIGHLIGHT, 27-120
HIGHNONE, 27-176
HOLDNMODIEY, 14-23
how, 27-543, 27-545
HP_LASERJET, 27-1641
HP_LASERJET_PLUS, 27-1642
HSIZEBITS, 3-29
HSIZEMASK, 3-31
ib_ActiveScreen, 28-51
ib_ActiveWindow, 28-50
ib_FirstScreen, 28-61
ib_LibNode, 28-48
ib_ViewLord, 28-49
icon, 23-31
IconDisp, 48-128
ICONNAME, 23-30
id_BytesPerBlock, 16-100
id_DiskState, 16-97
id_DiskType, 16-101
ID_DOS_DISK, 16-115
id_InUse, 16-103
ID_KICKSTART_DISK, 16-116
ID_NO_DISK_PRESENT, 16-112
ID_NOT_REALLY DOS, 16-114
id_NumBlocks, 16-98
id_NumBlocksUsed, 16-98
id_NumSoftErrors, 16-99
id_SIZEOF, 16-104
id_UnitNumber, 16-96
ID_UNREADABLE_DISK, 16-113
ID_VALIDATED, 16-110
ID_VALIDATING, 16-109
id_VolumeNode, 16-102
ID_WRITE_PROTECTED, 16-108
IDCMP, 27-940
ie_Class, 25-135
ie_Code, 25-137
ie_EventAddress, 25-139
ie_NextEvent, 25-134
ie_Qualifier, 25-138
ie_SIZEOF, 25-143
ie_SubClass, 25-136

```

ie\_TimeStamp, 25-142  
 ie\_X, 25-140  
 ie\_Y, 25-141  
 IECLASS\_ACTIVEWINDOW, 25-53  
 IECLASS\_CLOSEWINDOW, 25-41  
 IECLASS\_DISKINSERTED, 25-51  
 IECLASS\_DISKREMOVED, 25-49  
 IECLASS\_EVENT, 25-27  
 IECLASS\_GADGETDOWN, 25-33  
 IECLASS\_GADGETUP, 25-35  
 IECLASS\_INACTIVEWINDOW, 25-55  
 IECLASS\_MAX, 25-58  
 IECLASS\_MENULIST, 25-39  
 IECLASS\_NEWREFS, 25-47  
 IECLASS\_NULL, 25-21  
 IECLASS\_POINTERPOS, 25-29  
 IECLASS\_RAWKEY, 25-23  
 IECLASS\_RAWMOUSE, 25-25  
 IECLASS\_REFRESHWINDOW, 25-45  
 IECLASS\_REQUESTER, 25-37  
 IECLASS\_SIZEWINDOW, 25-43  
 IECLASS\_TIMER, 25-31  
 IECODE\_ASCII\_DEL, 25-74  
 IECODE\_ASCII\_FIRST, 25-72  
 IECODE\_ASCII\_LAST, 25-73  
 IECODE\_C0\_FIRST, 25-70  
 IECODE\_C0\_LAST, 25-71  
 IECODE\_C1\_FIRST, 25-75  
 IECODE\_C1\_LAST, 25-76  
 IECODE\_COMM\_CODE\_FIRST, 25-66  
 IECODE\_COMM\_CODE\_LAST, 25-67  
 IECODE\_KEY\_CODE\_FIRST, 25-64  
 IECODE\_KEY\_CODE\_LAST, 25-65  
 IECODE\_LATIN1\_FIRST, 25-77  
 IECODE\_LATIN1\_LAST, 25-78  
 IECODE\_LBUTTON, 25-81  
 IECODE\_MBUTTON, 25-83  
 IECODE\_NEWACTIVE, 25-87  
 IECODE\_NOBUTTON, 25-84  
 IECODE\_RBUTTON, 25-82  
 IECODE\_REOCLEAR, 25-94  
 IECODE\_REQSET, 25-92  
 IECODE\_UP\_PREFIX, 25-62  
 IECODEB\_UP\_PREFIX, 25-63  
 IEQUALIFIER\_CAPSLOCK, 25-102  
 IEQUALIFIER\_CONTROL, 25-104  
 IEQUALIFIER\_INTERRUPT, 25-118  
 IEQUALIFIER\_LALT, 25-106  
 IEQUALIFIER\_LBUTTON, 25-122  
 IEQUALIFIER\_LCOMMAND, 25-110  
 IEQUALIFIER\_LSHIFT, 25-98  
 IEQUALIFIER\_MBUTTON, 25-126  
 IEQUALIFIER\_MULTIBROADCAST, 25-120  
 IEQUALIFIER\_NUMERICPAD, 25-114  
 IEQUALIFIER\_RALT, 25-108  
 IEQUALIFIER\_RBUTTON, 25-124

IEQUALIFIER\_RCOMMAND, 25-112  
 IEQUALIFIER\_RELATIVEMOUSE, 25-128  
 IEQUALIFIER\_REPEAT, 25-116  
 IEQUALIFIER\_RSHIFT, 25-100  
 IEQUALIFIERB\_CAPSLOCK, 25-103  
 IEQUALIFIERB\_CONTROL, 25-105  
 IEQUALIFIERB\_INTERRUPT, 25-119  
 IEQUALIFIERB\_LALT, 25-107  
 IEQUALIFIERB\_LBUTTON, 25-123  
 IEQUALIFIERB\_LCOMMAND, 25-111  
 IEQUALIFIERB\_LSHIFT, 25-99  
 IEQUALIFIERB\_MBUTTON, 25-127  
 IEQUALIFIERB\_MULTIBROADCAST, 25-121  
 IEQUALIFIERB\_NUMERICPAD, 25-115  
 IEQUALIFIERB\_RALT, 25-109  
 IEQUALIFIERB\_RBUTTON, 25-125  
 IEQUALIFIERB\_RCOMMAND, 25-113  
 IEQUALIFIERB\_RELATIVEMOUSE, 25-129  
 IEQUALIFIERB\_REPEAT, 25-117  
 IEQUALIFIERB\_RSHIFT, 25-101  
 IEGT, 37-89  
 ig\_Depth, 27-769  
 ig\_Height, 27-768  
 ig\_ImageData, 27-770  
 ig\_LeftEdge, 27-763  
 ig\_NextImage, 27-830  
 ig\_PlaneOnOff, 27-823  
 ig\_PlanePick, 27-822  
 ig\_SIZEOF, 27-833, 48-60, 48-61  
 ig\_TopEdge, 27-765  
 ig\_Width, 27-767  
 ignored, 27-349  
 im\_Class, 27-855  
 im\_Code, 27-859  
 im\_ExecMessage, 27-846  
 im\_IAddress, 27-868  
 im\_IDCMPWindow, 27-888  
 im\_Micros, 27-883  
 im\_MouseX, 27-876  
 im\_MouseY, 27-877  
 im\_Qualifier, 27-863  
 im\_Seconds, 27-882  
 im\_SIZEOF, 27-893  
 im\_SpecialLink, 27-891  
 image, 27-175, 27-253, 27-363, 27-761, 27-778, 27-828  
 IMAGE\_NEGATIVE, 27-1610  
 IMAGE\_POSITIVE, 27-1609  
 imagery, 27-256, 27-272, 27-363, 27-782  
 INACTIVEWINDOW, 27-934  
 IND\_ADDHANDLER, 24-19  
 IND\_REMHANDLER, 24-20  
 IND\_SETMPORT, 24-24  
 IND\_SETMTRIC, 24-26  
 IND\_SETMTYPE, 24-25  
 IND\_SETPERIOD, 24-23  
 IND\_SETTHRESH, 24-22

IND\_WRITEEVENT, 24-21  
     Info, 17-32  
     InfoData, 16-94  
     InitAnimate, 20-49  
     Initialize, 27-1259  
     inner-Window, 27-1095  
     Input, 17-22  
     InputEvent, 25-133, 27-61  
     INREQUEST, 27-1154  
     INTB\_AUDIO, 26-26  
     INTB\_AUD1, 26-25  
     INTB\_AUD2, 26-24  
     INTB\_AUD3, 26-23  
     INTB\_BLIT, 26-27  
     INTB\_COPER, 26-29  
     INTB\_DSKBLK, 26-32  
     INTB\_DSKSYNC, 26-21  
     INTB\_EXTER, 26-20  
     INTB\_INTEN, 26-19  
     INTB\_PORTS, 26-30  
     INTB\_RBE, 26-22  
     INTB\_SETCLR, 26-16  
     INTB\_SOFTINT, 26-31  
     INTB\_TBE, 26-33  
     INTB\_VERTB, 26-28  
     integer, 27-637  
     intena, 11-85  
     intena, 11-34  
     INTERLACE, 14-24  
     interrupts, 12-33, 22-16  
     INTE\_AUDIO, 26-45  
     INTE\_AUD1, 26-44  
     INTE\_AUD2, 26-43  
     INTE\_AUD3, 26-42  
     INTE\_BLIT, 26-46  
     INTE\_COPER, 26-48  
     INTE\_DSKBLK, 26-51  
     INTE\_DSKSYNC, 26-40  
     INTE\_EXTER, 26-39  
     INTE\_INTEN, 26-38  
     INTE\_PORTS, 26-49  
     INTE\_RBE, 26-41  
     INTE\_SETCLR, 26-37  
     INTE\_SOFTINT, 26-50  
     INTE\_TBE, 26-52  
     INTE\_VERTB, 26-47  
     intreq, 11-86  
     intreq, 11-35  
     IntuiMessage, 27-844  
     Intuit, 27-349  
     IntuiText, 27-670  
     INTUITICKS, 27-938  
     IntuitionBase, 28-46, 28-53  
     io\_Actual, 8-48  
     IO\_BAUD, 40-117  
     IO\_BRKTIME, 40-118

    io\_ClipID, 8-52  
     io\_ColorMap, 36-139  
     io\_Command, 8-45  
     IO\_CTLCHAR, 40-114  
     io\_Data, 8-50  
     io\_DestCols, 36-145  
     io\_DestRows, 36-146  
     io\_Device, 8-43  
     io\_Error, 8-47  
     IO\_EXTFLAGS, 40-116  
     io\_Flags, 8-46  
     io\_Length, 8-49  
     io\_Message, 8-42  
     io\_Modes, 36-140  
     io\_Offset, 8-51  
     IO\_PAREFLAGS, 34-121  
     io\_Parm0, 36-131  
     io\_Parm1, 36-132  
     io\_Parm2, 36-133  
     io\_Parm3, 36-134  
     IO\_PARSTATUS, 34-120  
     IO\_PEXTFLAGS, 34-119  
     io\_PrtCommand, 36-130  
     IO\_PTERMARRAY, 34-122  
     io\_RastPort, 36-138  
     IO\_RBUFLen, 40-115  
     IO\_READLEN, 40-120  
     IO\_SEREFLAGS, 40-123  
     IO\_SIZE, 2-44, 36-129, 36-137, 44-36  
     io\_Special, 36-147  
     io\_SrcHeight, 36-144  
     io\_SrcWidth, 36-143  
     io\_SrcX, 36-141  
     io\_SrcY, 36-142  
     IO\_STATUS, 40-124  
     IO\_STOPBITS, 40-122  
     IO\_TERMARRAY, 40-119  
     io\_Unit, 8-44  
     IO\_WRITELEN, 40-121  
     ioa\_AllocKey, 2-45  
     ioa\_Cycles, 2-50  
     ioa\_Data, 2-46  
     ioa\_Length, 2-47  
     ioa\_Period, 2-48  
     ioa\_SIZEOF, 2-52  
     ioa\_Volume, 2-49  
     ioa\_WriteMsg, 2-51  
     IOAudio, 2-44  
     IOClipReq, 8-41  
     ioCr\_SIZEOF, 8-53  
     iodrpr\_SIZEOF, 36-148  
     IODRPRReq, 36-137  
     IoErr, 17-35  
     IOEXTPAR, 34-95  
     IOEXTPar\_SIZE, 34-123, 37-90, 37-91  
     IOEXTPar\_SIZE-IOEXTSER\_SIZE, 37-89, 37-94

IOEXTSER, 40-91  
 IOEXTSER\_SIZE, 37-95, 37-96, 40-142  
 IOEXTTD, 45-105  
 IOPAR, 34-74, 34-75, 34-76  
 iopcr\_SIZEOF, 36-135  
 IOPrtCmdReq, 36-129  
 IOPT, 34-77, 34-78, 34-79, 34-80  
 IORO, 37-76  
 IORL, 37-77  
 IOSER, 40-71, 40-72, 40-73  
 IOST, 40-74, 40-75, 40-76, 40-77, 40-78  
 IOSTD\_SIZE, 33-58, 34-95, 40-91, 45-105  
 IOTD\_COUNT, 45-106  
 IOTD\_SECLABEL, 45-107  
 IOTD\_SIZE, 45-108  
 IOTV\_SIZE, 37-99, 44-38  
 IOTV\_TIME, 44-37  
 IS\_SIZE, 6-23, 12-48, 12-49, 12-50, 12-63, 12-64,  
 12-65, 22-30, 22-31, 22-32  
 ISDRAWN, 27-183  
 ISGRTRX, 7-68  
 ISGRTRY, 7-69  
 IsInteractive, 17-49  
 ISLESSX, 7-66  
 ISLESSY, 7-67  
 it, 27-337, 27-389, 27-1724  
 it\_BackPen, 27-674  
 it\_DrawMode, 27-676  
 it\_FrontPen, 27-672  
 it\_IText, 27-706  
 it\_ITextFont, 27-704  
 it\_KludgeFill00, 27-696  
 it\_LeftEdge, 27-699  
 it\_NextText, 27-708  
 it\_SIZEOF, 27-711  
 it\_TopEdge, 27-701  
 ITALIC, 43-20  
 item, 27-115, 27-158, 27-165  
 ITEMENABLED, 27-166  
 items, 27-120  
 ITEMTEXT, 27-157  
 IText, 27-1732  
 its, 27-1157  
 IV\_SIZE, 6-24, 6-25, 6-26, 6-27, 6-28  
 joy0dat, 11-24  
 joyldat, 11-25  
 joytest, 11-47  
 K, 4-48, 4-48, 16-116, 16-116  
 KBD\_ADDRESSHANDLER, 29-21  
 KBD\_READEVENT, 29-19  
 KBD\_READMATRIX, 29-20  
 KBD\_REMRESETHANDLER, 29-22  
 KBD\_RESETHANDLERDONE, 29-23  
 KC\_NOQUAL, 30-28  
 KC\_VANILLA, 30-29  
 KCB\_CONTROL, 30-32

KCB\_DOWNUP, 30-34  
 KCB\_NOP, 30-25  
 KCB\_STRING, 30-37  
 KCF\_ALT, 30-31  
 KCF\_CONTROL, 30-33  
 KCF\_DOWNUP, 30-35  
 KCF\_NOP, 30-26  
 KCF\_SHIFT, 30-30  
 KCF\_STRING, 30-38  
 KeyMap, 30-13  
 KICK, 4-43  
 km\_HiCapsable, 30-20  
 km\_HiKeyMap, 30-19  
 km\_HiKeyMapTypes, 30-18  
 km\_HiRepeatable, 30-21  
 km\_LoCapsable, 30-16  
 km\_LoKeyMap, 30-15  
 km\_LoKeyMapTypes, 30-14  
 km\_LoRepeatable, 30-17  
 km\_SIZEOF, 30-22  
 KNOBHIT, 27-590  
 KNOBHMIN, 27-593  
 KNOBVMIN, 27-594  
 Layer, 7-15  
 Layer\_Info, 31-32  
 LayerInfo, 27-1357  
 LayerInfo\_extra, 31-16  
 layers, 27-41  
 leaves, 27-337  
 left, 27-380  
 LEFTBORDER, 27-451  
 LeftTop, 27-741  
 LETTER, 27-1602  
 li\_broadcast, 31-39  
 li\_bytereserved, 31-43  
 li\_check\_lp, 31-34  
 li\_LayerInfo\_extra, 31-46  
 li\_Lock, 31-38  
 li\_Locker, 31-42  
 li\_locknest, 31-40  
 li\_LockPort, 31-37  
 li\_longreserved, 31-45  
 li\_obs, 31-35  
 li\_pad, 31-41  
 li\_RP\_ReplyPort, 31-36  
 li\_SIZEOF, 27-1356, 31-47  
 li\_top\_layer, 31-33  
 li\_wordreserved, 31-44  
 LIB\_BASE, 12-105, 32-48  
 LIB\_SIZE, 6-18, 12-55, 18-150, 22-19, 28-48, 32-44, 37-47  
 LIBINIT, 12-105, 32-48  
 library, 16-16, 23-31  
 lie\_blitbuff, 31-20  
 lie\_env, 31-17  
 lie\_FreeClipRects, 31-19  
 lie\_mem, 31-18



```

lie_SIZEOF, 31-21
 like, 27-866
LINEMODE, 3-67
 list, 27-217
 listing, 27-53, 27-106, 27-159, 27-212, 27-265,
 27-318, 27-372, 27-425, 27-478, 27-531,
 27-584, 27-638, 27-691, 27-744, 27-798,
 27-851, 27-905, 27-959, 27-1012, 27-1065,
 27-1118, 27-1171, 27-1225, 27-1278, 27-1331,
 27-1384, 27-1437, 27-1491, 27-1544, 27-1597,
 27-1650, 27-1704, 28-53
LMN_REGION, 31-23
 LN_PRI, 37-59
 LoadSeg, 17-38
 location, 27-666
 Lock, 17-27
LONELYMESSAGE, 27-950
 LONGINT, 27-461
LOWCHECKWIDTH, 27-1711
LOWCOMMWIDTH, 27-1712
lr_cliprects, 7-40
 lr_pl, 7-47
 lr_Back, 7-17
 lr_ClipRect, 7-18
 lr_cr, 7-44
 lr_cr2, 7-45
 lr_crnew, 7-46
 lr_DamageList, 7-39
 lr_Flags, 7-29
 lr_Front, 7-16
 lr_l_LockMessage, 7-38
 lr_LayerInfo, 7-41
 lr_LayerLockCount, 7-26
 lr_LayerLocker, 7-42
 lr_Lock, 7-24
 lr_LockCount, 7-25
 lr_LockMessage, 7-36
 lr_LockPort, 7-35
 lr_MaxX, 7-22
 lr_MaxY, 7-23
 lr_MinX, 7-20
 lr_MinY, 7-21
 lr_RastPort, 7-19
 lr_ReplyPort, 7-37
 lr_reserved, 7-27
 lr_reserved1, 7-28
 lr_Scroll_X, 7-33
 lr_Scroll_Y, 7-34
 lr_SIZEOF, 7-48
 lr_SuperBitMap, 7-30
 lr_SuperClipRect, 7-31
 lr_SuperSaverClipRects, 7-43
 lr_Window, 7-32
 M_ASM, 9-95
 M_AWM, 9-98
 M_LNM, 9-94

```

```

MALE, 33-20, 33-22
MAXBODY, 27-595
MAXBYTESPERROW, 3-34
MAXCYLS, 45-33
MAXFONTNAME, 13-41, 13-55
MAXFONTPATH, 13-23, 13-26
MAXFREQ, 33-32
MAXINT, 16-37
MAXPITCH, 33-30
MAXPOT, 27-596
MAXRATE, 33-28
MAXRETRY, 45-37
MAXVOL, 33-34
 me, 27-949
 means, 27-525
 mem_node, 31-25
 memnode_how_big, 31-29
 memnode_pred, 31-27
 memnode_SIZEOF, 31-30
 memnode_succ, 31-26
 memnode_where, 31-28
 MEMORY, 13-60
 Menu, 27-68
 MENUCANCEL, 27-963
 MENUENABLED, 27-89
 MENUHOT, 27-957
 MenuItem, 27-101
 MENUNULL, 27-1691
 MENUPICK, 27-915
 Menus, 27-1157
 MENUSTATE, 27-1156
 MENUTOGGLE, 27-164
 MENUTOGGLED, 27-186
 MENUVERIFY, 27-923
 MENUWAITING, 27-965
 mi_Command, 27-126
 mi_Flags, 27-112
 mi_Height, 27-111
 mi_ItemFill, 27-117
 mi_KludgeFill100, 27-142
 mi_LeftEdge, 27-104
 mi_MutualExclude, 27-114
 mi_NextItem, 27-103
 mi_NextSelect, 27-150
 mi_SelectFill, 27-123
 mi_SIZEOF, 27-152
 mi_SubItem, 27-145
 mi_TopEdge, 27-105
 mi_Width, 27-110
 Micros, 27-880
 MIDRAWN, 27-92
 MINREQ, 33-31
 MININT, 16-38
 MINPITCH, 33-29
 MINRATE, 33-27
 MINVOL, 33-33

```

MinWidth, 27-1263  
     misc, 32-54  
     MISCNAME, 32-53  
 MiscResource, 32-44  
     mode, 27-456  
     MODE\_640, 14-16  
     MODE\_NEWFILE, 16-24  
     MODE\_OLDFILE, 16-22  
     MOUSEBUTTONS, 27-903  
     MOUSEMOVE, 27-909  
     MR\_SIZE, 7-35, 7-37, 18-35, 31-36, 31-37, 37-81, 37-100  
     mr\_AllocArray, 32-45  
 MR\_ALLOCMISRESOURCE, 32-49  
 MR\_FREEMISRESOURCE, 32-50  
     MR\_PARALLELBITS, 32-40  
     MR\_PARALLELPORT, 32-39  
     MR\_SERIALBITS, 32-38  
     MR\_SERIALPORT, 32-37  
     mr\_Sizeof, 32-46  
     MRB, 33-75  
     MRB\_HEIGHT, 33-77  
     MRB\_PAD, 33-79  
     MRB\_SHAPE, 33-78  
     MRB\_SIZE, 33-80  
     MRB\_WIDTH, 33-76  
 MTYPE\_CLOSEDOWN, 48-142  
 MTYPE\_DISKCHANGE, 48-140  
     MTYPE\_IOPROC, 48-143  
     MTYPE\_PSTD, 48-138  
     MTYPE\_TIMER, 48-141  
     MTYPE\_TOOLEXIT, 48-139  
     mu\_BeatX, 27-83  
     mu\_BeatY, 27-84  
     mu\_FirstItem, 27-77  
     mu\_Flags, 27-75  
     mu\_Height, 27-74  
     mu\_JazzX, 27-81  
     mu\_JazzY, 27-82  
     mu\_LeftEdge, 27-71  
     mu\_MenuName, 27-76  
     mu\_NextMenu, 27-70  
     mu\_SIZEOF, 27-86  
     mu\_TopEdge, 27-72  
     mu\_Width, 27-73  
     MUSTDRAW, 20-17  
     N, 16-114  
     N\_TRACTOR, 27-1624  
     NABC, 3-42  
     NABNC, 3-43  
     NANBC, 3-44  
     NANBNC, 3-45  
     NATURALFO, 33-18, 33-23  
     ND\_CantAlloc, 33-43  
     ND\_Expunged, 33-46  
     ND\_FreqErr, 33-52  
     ND\_MakeBad, 33-41

    ND\_ModeErr, 33-51  
     ND\_NoAudLib, 33-40  
     ND\_NoMem, 33-39  
     ND\_NotUsed, 33-38  
     ND\_NoWrite, 33-45  
     ND\_PhonErr, 33-47  
     ND\_PitchErr, 33-49  
     ND\_RateErr, 33-48  
     ND\_SexErr, 33-50  
     ND\_Unimpl, 33-44  
     ND\_UnitErr, 33-42  
     ND\_VolErr, 33-53  
     NDI, 33-58  
     NDI\_CHANMASK, 33-68  
     NDI\_CHMASKS, 33-63  
     NDI\_MODE, 33-61  
     NDI\_MOUTHS, 33-67  
     NDI\_NUMCHAN, 33-69  
     NDI\_NUMMASKS, 33-64  
     NDI\_PAD, 33-70  
     NDI\_PITCH, 33-60  
     NDI\_RATE, 33-59  
     NDI\_SAMPREQ, 33-66  
     NDI\_SEX, 33-62  
     NDI\_SIZE, 33-71, 33-75  
     NDI\_VOLUME, 33-65  
 NEWLAYERINFO\_CALLED, 31-49  
     NEWPREFS, 27-924  
     NewScreen, 27-1415  
     NEWSIZE, 27-900  
     NewWindow, 27-1197  
     next, 27-148  
     no, 27-819  
 NO\_ICON\_POSITION, 48-168  
     NOCAREREFRESH, 27-1162  
     NOCROSSEFILL, 38-47  
     nodes, 8-14, 13-14, 36-20, 37-15, 48-19  
     NOITEM, 27-1689  
     NOMENU, 27-1688  
     NOSUB, 27-1690  
     not, 27-397  
     notwithstanding, 27-875  
     ns\_BlockPen, 27-1425  
     ns\_DefaultTitle, 27-1434  
     ns\_Depth, 27-1421  
     ns\_DetailPen, 27-1423  
     ns\_Font, 27-1432  
     ns\_Gadgets, 27-1436  
     ns\_Height, 27-1420  
     ns\_LeftEdge, 27-1417  
     ns\_SIZEOF, 27-1454  
     ns\_TopEdge, 27-1418  
     ns\_Type, 27-1430  
     ns\_ViewModes, 27-1428  
     ns\_Width, 27-1419  
     NULL, 27-124, 27-307, 27-619

NUMCYLS, 45-32, 45-33  
 NUMHEADS, 45-36  
 NUMMRYPES, 32-42  
 NUMSECS, 45-35  
 NUMTRACKS, 45-38  
 NUMUNITS, 45-39  
 nw\_BitMap, 27-1248  
 nw\_BlockPen, 27-1206  
 nw\_CheckMark, 27-1230  
 nw\_DetailPen, 27-1204  
 nw\_FirstGadget, 27-1219  
 nw\_Flags, 27-1211  
 nw\_Height, 27-1202  
 nw\_IDCMPFlags, 27-1209  
 nw\_LeftEdge, 27-1199  
 nw\_MaxHeight, 27-1272  
 nw\_MaxWidth, 27-1271  
 nw\_MinHeight, 27-1270  
 nw\_MinWidth, 27-1269  
 nw\_Screen, 27-1241  
 nw\_SIZE, 27-1287, 48-47, 48-71  
 nw\_Title, 27-1232  
 nw\_TopEdge, 27-1200  
 nw\_Type, 27-1285  
 nw\_Width, 27-1201  
 O, 4-47, 16-114, 16-115  
 OCTANT1, 3-87  
 OCTANT2, 3-86  
 OCTANT3, 3-85  
 OCTANT4, 3-84  
 OCTANT5, 3-83  
 OCTANT6, 3-82  
 OCTANT7, 3-81  
 OCTANTS, 3-80  
 OFFSET\_BEGINING, 16-32  
 OFFSET\_BEGINNING, 16-28, 16-32  
 OFFSET\_CURRENT, 16-29  
 OFFSET\_END, 16-30  
 offsets, 27-208, 27-210  
 offwindow, 27-276  
 OK, 27-251  
 OKIMATE\_20, 27-1638  
 on, 27-1157  
 one, 27-1149, 27-1277, 27-1718  
 ONE\_DOT, 38-43, 38-105  
 ONE\_DOTn, 38-104  
 ONEDOT, 3-71  
 only, 27-80, 27-780  
 Open, 17-18  
 operation, 27-964  
 option, 27-1253  
 OR, 20-55, 27-397, 27-418  
 order, 27-136, 27-233, 27-686, 27-1340  
 OTHER\_REFRESH, 27-1135  
 Output, 17-23  
 OUTSTEP, 20-36

over, 27-404, 27-1031  
 OVERLAY, 20-16  
 OVERRUN, 40-78  
 OVELAG, 3-72  
 OWNBLITTERn, 22-55  
 P, 37-76, 37-77, 37-78  
 P\_PRIORITY, 37-72  
 P\_STKSIZE, 37-73, 37-102  
 PAPEROUT, 34-79  
 PAR, 34-71, 34-72, 34-73  
 Par\_DEVEINISH, 34-50  
 parallel, 34-68, 37-31  
 PARALLEL\_PRINTER, 27-1574  
 PARALLELNAME, 34-67  
 ParentDir, 17-48  
 ParErr\_BuffTooBig, 34-35  
 ParErr\_DevBusy, 34-34  
 ParErr\_InitErr, 34-40  
 ParErr\_InvParam, 34-36  
 ParErr\_LineErr, 34-37  
 ParErr\_NotOpen, 34-38  
 ParErr\_PortReset, 34-39  
 PARTY\_ODD, 40-69  
 PARTY\_ON, 40-70  
 PBUSY, 34-78  
 PCC\_BW, 37-116  
 PCC\_YMC, 37-117  
 PCC\_YMC\_BW, 37-118  
 PCC\_YMCB, 37-119  
 pd\_Flags, 37-103  
 pd\_IOR0, 37-90, 37-95  
 pd\_IOR1, 37-91, 37-96  
 pd\_IORPort, 37-100  
 pd\_pad, 37-104  
 pd\_PBothReady, 37-87  
 pd\_Preferences, 37-105  
 pd\_PrintBuf, 37-85  
 pd\_PrinterSegment, 37-82  
 pd\_PrinterType, 37-83  
 pd\_PWaitEnabled, 37-106  
 pd\_PWrite, 37-86  
 pd\_SegmentData, 37-84  
 pd\_SIZEOF, 37-107  
 pd\_Stk, 37-102  
 pd\_TC, 37-101  
 pd\_TIOR, 37-99  
 pd\_Unit, 37-81  
 PDCMD\_QUERY, 34-48  
 PDCMD\_SETPARAMS, 34-49  
 PDERR\_BADDIMENSION, 36-166  
 PDERR\_BUFFERMEMORY, 36-169  
 PDERR\_CANCEL, 36-163  
 PDERR\_DIMENSIONOVERFLOW, 36-167  
 PDERR\_INTERNALMEMORY, 36-168  
 PDERR\_INVERTHAM, 36-165  
 PDERR\_NOTGRAPHICS, 36-164

ped\_Close, 37-126  
ped\_ColorClass, 37-128  
ped\_Commands, 37-136  
ped\_DoSpecial, 37-137  
ped\_Expunge, 37-124  
ped\_Init, 37-123  
ped\_MaxColumns, 37-129  
ped\_MaxXDots, 37-132  
ped\_MaxYDots, 37-133  
ped\_NumCharSets, 37-130  
ped\_NumRows, 37-131  
ped\_Open, 37-125  
ped\_PrinterClass, 37-127  
ped\_PrinterName, 37-122  
ped\_Render, 37-138  
ped\_SIZEOF, 37-140  
ped\_TimeoutSecs, 37-139  
ped\_XDotsInch, 37-134  
ped\_YDotsInch, 37-135  
percentage, 27-537  
PF2PRI, 14-20  
pf\_BaudRate, 27-1509  
pf\_color0, 27-1528  
pf\_color1, 27-1529  
pf\_color17, 27-1522  
pf\_color18, 27-1523  
pf\_color19, 27-1524  
pf\_color2, 27-1530  
pf\_color3, 27-1531  
pf\_DoubleClick, 27-1514  
PF\_FINE\_SCROLL\_MASK, 14-29  
pf\_FontHeight, 27-1503  
pf\_KeyRptDelay, 27-1513  
pf\_KeyRptSpeed, 27-1512  
pf\_padding, 27-1566  
pf\_PaperLength, 27-1563  
pf\_PaperSize, 27-1562  
pf\_PaperType, 27-1564  
pf\_PointerMatrix, 27-1518  
pf\_PointerTicks, 27-1525  
pf\_PrintAspect, 27-1556  
pf\_PrinterFilename, 27-1543  
pf\_PrinterPort, 27-1506  
pf\_PrinterType, 27-1542  
pf\_PrintImage, 27-1555  
pf\_PrintLeftMargin, 27-1553  
pf\_PrintPitch, 27-1550  
pf\_PrintQuality, 27-1551  
pf\_PrintRightMargin, 27-1554  
pf\_PrintShade, 27-1557  
pf\_PrintSpacing, 27-1552  
pf\_PrintThreshold, 27-1558  
pf\_SIZEOF, 27-1568, 37-105  
pf\_ViewInitX, 27-1536  
pf\_ViewInitY, 27-1537  
pf\_ViewXOffset, 27-1534

pf\_ViewYOffset, 27-1535  
pf\_XOffset, 27-1520  
pf\_YOffset, 27-1521  
PEA\_FINE\_SCROLL, 14-27  
PEB\_FINE\_SCROLL\_SHIFT, 14-28  
pi\_Height, 27-573  
pi\_CWidth, 27-571  
pi\_Flags, 27-511  
pi\_HorizBody, 27-567  
pi\_HorizPot, 27-527  
pi\_HPOTRes, 27-575  
pi\_LeftBorder, 27-577  
pi\_SIZEOF, 27-579, 48-62, 48-63  
pi\_TopBorder, 27-578  
pi\_VertBody, 27-568  
pi\_VertPot, 27-529  
pi\_VPOTRes, 27-576  
PICA, 27-1592  
plane, 27-788  
planes, 27-780, 27-819  
PLNCNTMSK, 14-17  
PLNCNTSHET, 14-19  
pointer, 27-270, 27-1519  
POINTERSIZE, 27-1474  
POINTREL, 27-269  
pot0dat, 11-29  
pot1dat, 11-30  
potgo, 11-46, 35-8  
POTGONAME, 35-7  
potinp, 11-31  
PPC, 37-109, 37-110  
PPC\_BWALPHA, 37-112  
PPC\_BWGEX, 37-113  
PPC\_COLORGEX, 37-114  
pr\_CIS, 18-44  
pr\_CLI, 18-48  
pr\_ConsoleTask, 18-46  
pr\_COS, 18-45  
pr\_CurrentDir, 18-43  
pr\_FileSystemTask, 18-47  
pr\_GlobVec, 18-39  
pr\_MsgPort, 18-35  
pr\_Pad, 18-36  
pr\_PktWait, 18-50  
pr\_Result2, 18-42  
pr\_ReturnAddr, 18-49  
pr\_SegList, 18-37  
pr\_SIZEOF, 18-52  
pr\_StackBase, 18-41  
pr\_StackSize, 18-38  
pr\_Task, 18-34  
pr\_TaskNum, 18-40  
pr\_WindowPtr, 18-51  
PRD\_DUMPREPORT, 36-39  
PRD\_PRTCOMMAND, 36-38  
PRD\_RAWWRITE, 36-37

PREDRAWN, 27-271  
 Preferences, 27-1500  
 PRIMARY\_CLIP, 8-57  
 PrinterData, 37-80  
 PrinterExtendedData, 37-121  
 PrinterSegment, 37-142  
 Process, 18-33  
 PROPBORDERLESS, 27-589  
 PROPGADGET, 27-494  
 PropInfo, 27-509  
 Proportional, 27-343, 43-30  
 ps\_NextSegment, 37-143  
 ps\_PED, 37-147  
 ps\_Revision, 37-146  
 ps\_runAlert, 37-144  
 ps\_Version, 37-145  
 PSEL, 34-80  
 PTERMARRAY, 34-85  
 PTERMARRAY\_0, 34-86  
 PTERMARRAY\_1, 34-87  
 PTERMARRAY\_SIZE, 34-88, 34-122  
 QBOWNER, 22-58  
 QBOWNERn, 22-56, 22-58  
 Qualifier, 27-862  
 QUEUED, 34-74, 37-62, 40-71  
 QUEUEDBRK, 40-67  
 QueuePacket, 17-41  
 QUME\_LP\_20, 27-1639  
 R, 18-122  
 ra\_MaxX, 21-27  
 ra\_MaxY, 21-28  
 ra\_MinX, 21-25  
 ra\_MinY, 21-26  
 ra\_SIZEOF, 21-29, 39-14, 39-21  
 RAD\_BOOGIE, 34-72, 40-66  
 RasInfo, 47-60  
 rastport, 27-37, 38-58  
 RAWKEY, 27-918  
 READ, 16-71, 17-20  
 READBREAK, 40-76  
 receive, 27-424  
 RECOVERY\_ALERT, 27-1721  
 Rectangle, 21-24, 27-817, 27-821  
 reflects, 27-1484  
 refptr, 11-40  
 REFRESH, 27-1163  
 REFRESHBITS, 27-1131  
 REFRESHWINDOW, 27-901  
 Region, 39-13  
 RegionRectangle, 39-18  
 rel, 27-378  
 relative, 27-873  
 RELVERIFY, 27-406  
 RemBob, 20-54  
 Remember, 27-1659  
 REMOVED, 43-32

Rename, 17-26  
 render, 27-723  
 REPORTMOUSE, 27-1140  
 REACTIVE, 27-277  
 REACTIVE, 27-277  
 REOCLEAR, 27-921  
 REOGADGET, 27-475  
 REQOFFWINDOW, 27-275  
 REQSET, 27-914  
 Requester, 27-197, 27-200  
 Requesters, 27-418, 27-1005  
 REVERIFY, 27-920  
 reserve, 17-8  
 reserved, 27-244, 27-260  
 resource, 5-7, 5-11, 12-101, 32-54, 35-8  
 restored, 27-434  
 RETURN\_ERROR, 16-146  
 RETURN\_FAIL, 16-147  
 RETURN\_OK, 16-144  
 RETURN\_WARN, 16-145  
 REVPATH, 43-27  
 rg\_bounds, 39-14  
 rg\_RegionRectangle, 39-15  
 rg\_SIZEOF, 39-16  
 ri\_BitMap, 47-62  
 ri\_Next, 47-61  
 ri\_RxOffset, 47-63  
 ri\_RyOffset, 47-64  
 ri\_SIZEOF, 47-65  
 RICHTBORDER, 27-450  
 RINGTRIGGER, 20-43  
 rm\_Memory, 27-1663  
 rm\_NextRemember, 27-1661  
 rm\_RememberSize, 27-1662  
 rm\_SIZEOF, 27-1665  
 RMBTRAP, 27-1160  
 rn\_ConsoleSegment, 18-164  
 rn\_Info, 18-167  
 rn\_RestartSeg, 18-166  
 rn\_SIZEOF, 18-168  
 rn\_TaskArray, 18-161  
 rn\_Time, 18-165  
 ROBOTIC0, 33-19  
 ROMEONT, 43-25  
 RootNode, 18-160  
 RP, 38-42, 38-43, 38-44, 38-46, 38-47, 38-56  
 rp\_AlgoStyle, 38-81  
 rp\_AOLPen, 38-68  
 rp\_AreaInfo, 38-63  
 rp\_AreaPtrn, 38-61  
 rp\_AreaPtSz, 38-70  
 rp\_BgPen, 38-67  
 rp\_BitMap, 38-60  
 RP\_COMPLEMENT, 38-52  
 rp\_cp\_x, 38-75  
 rp\_cp\_y, 38-76  
 rp\_DrawMode, 38-69

rp\_Dummy, 38-71  
 rp\_EgPen, 38-66  
 rp\_Flags, 38-73  
 rp\_Font, 38-80  
 rp\_GelsInfo, 38-64  
 RP\_INVERSVID, 38-53  
 RP\_JAM1, 38-50  
 RP\_JAM2, 27-1736, 38-51  
 rp\_Layer, 38-59  
 rp\_LinePtrn, 38-74  
 rp\_linpatcnt, 38-72  
 rp\_longreserved, 38-89  
 rp\_Mask, 38-65  
 rp\_minterms, 38-77  
 rp\_PenHeight, 38-79  
 rp\_PenWidth, 38-78  
 rp\_reserved, 38-90  
 rp\_RP\_User, 38-87  
 rp\_SIZEOF, 27-1353, 38-91  
 rp\_TmpRas, 38-62  
 rp\_TxBaseline, 38-85  
 rp\_TxFlags, 38-82  
 rp\_TxHeight, 38-83  
 rp\_TxSpacing, 38-86  
 rp\_TxWidth, 38-84  
 rp\_wordreserved, 38-88  
 rq\_BackFill, 27-223  
 rq\_Flags, 27-221  
 rq\_Height, 27-205  
 rq\_KludgeFill00, 27-239  
 rq\_LeftEdge, 27-202  
 rq\_OlderRequest, 27-201  
 rq\_RelLeft, 27-207  
 rq\_RelTop, 27-209  
 rq\_ReqBMap, 27-255  
 rq\_ReqBorder, 27-218  
 rq\_ReqGadget, 27-216  
 rq\_ReqLayer, 27-242  
 rq\_ReqPad1, 27-243  
 rq\_ReqPad2, 27-259  
 rq\_ReqText, 27-219  
 rq\_RWindow, 27-258  
 rq\_SIZEOF, 27-262  
 rq\_TopEdge, 27-203  
 rq\_Width, 27-204  
 rr\_bounds, 39-21  
 rr\_Next, 39-19  
 rr\_Rev, 39-20  
 rr\_SIZEOF, 39-22  
 RWDIR, 34-77  
 S, 4-47, 16-114, 16-115, 27-307, 27-368  
 same, 27-774  
 SatisfyMsg, 8-59  
 satisfyMsg\_SIZEOF, 8-63  
 save, 27-1025  
 SAVEBACK, 20-15

SAVEBOB, 20-28  
 SAVEPRESERVE, 20-35  
 sc\_BarHBorder, 27-1321  
 sc\_BarHeight, 27-1319  
 sc\_BarVBorder, 27-1320  
 sc\_BitMap, 27-1354  
 sc\_BlockPen, 27-1366  
 sc\_DefaultTitle, 27-1315  
 sc\_DetailPen, 27-1365  
 sc\_ExtData, 27-1376  
 sc\_FirstGadget, 27-1363  
 sc\_FirstWindow, 27-1301  
 sc\_Flags, 27-1312  
 sc\_Font, 27-1350  
 sc\_Height, 27-1307  
 sc\_KludgeFill00, 27-1346  
 sc\_LayerInfo, 27-1356  
 sc\_LeftEdge, 27-1303  
 sc\_MenuHBorder, 27-1323  
 sc\_MenuVBorder, 27-1322  
 sc\_MouseX, 27-1310  
 sc\_MouseY, 27-1309  
 sc\_NextScreen, 27-1300  
 sc\_RastPort, 27-1353  
 sc\_SaveColor0, 27-1371  
 sc\_SIZEOF, 27-1381  
 sc\_Title, 27-1314  
 sc\_TopEdge, 27-1304  
 sc\_UserData, 27-1378  
 sc\_ViewPort, 27-1351  
 sc\_WBorBottom, 27-1327  
 sc\_WBorLeft, 27-1325  
 sc\_WBorRight, 27-1326  
 sc\_WBorTop, 27-1324  
 sc\_Width, 27-1306  
 screen, 27-382, 27-385, 27-1298, 27-1318, 27-1390, 27-1449  
 Screens, 28-58  
 SCREENTYPE, 27-1392  
 SCRGADGET, 27-475  
 sd\_ctl, 11-118  
 sd\_dataa, 11-119  
 sd\_datab, 11-120  
 sd\_pos, 11-117  
 SDCMD\_BREAK, 40-43  
 SDCMD\_QUERY, 40-42  
 SDCMD\_SETPARAMS, 40-44  
 SDOWNBACK, 27-489  
 SDRAGGING, 27-485  
 Seek, 17-24  
 selected, 27-148, 27-156, 27-391, 48-130  
 selectors, 27-784  
 SER, 40-63, 40-64, 40-65, 40-66, 40-67, 40-68,  
 40-69, 40-70  
 SER\_CTL, 40-34  
 SER\_DBAUD, 40-35  
 SER\_DEVEFINISH, 40-46

```

 serdat, 11-44
 serdatr, 11-32
SerErr_BaudMismatch, 40-153
SerErr_BufErr, 40-155
SerErr_BufOverflow, 40-163
SerErr_DetectedBreak, 40-166
SerErr_DevBusy, 40-152
SerErr_InitErr, 40-161
SerErr_InvBaud, 40-154
SerErr_InvParam, 40-156
SerErr_LineErr, 40-157
SerErr_NoCTS, 40-165
SerErr_NoDSR, 40-164
SerErr_NotOpen, 40-158
SerErr_ParityErr, 40-160
SerErr_PortReset, 40-159
SerErr_TimerErr, 40-162
 serial, 37-34, 40-60
SERIAL_PRINTER, 27-1575
SERIALNAME, 40-59
 serper, 11-45
SERVICING, 37-64
SetComment, 17-43
SetProtection, 17-44
 settings, 27-517, 27-523
 SGR_BLACK, 9-39
 SGR_BLACKBG, 9-49
 SGR_BLUE, 9-43
 SGR_BLUEBG, 9-53
 SGR_BOLD, 9-33
 SGR_CLRO, 9-61
 SGR_CLR0BG, 9-70
 SGR_CLR1, 9-62
 SGR_CLR1BG, 9-71
 SGR_CLR2, 9-63
 SGR_CLR2BG, 9-72
 SGR_CLR3, 9-64
 SGR_CLR3BG, 9-73
 SGR_CLR4, 9-65
 SGR_CLR4BG, 9-74
 SGR_CLR5, 9-66
 SGR_CLR5BG, 9-75
 SGR_CLR6, 9-67
 SGR_CLR6BG, 9-76
 SGR_CLR7, 9-68
 SGR_CLR7BG, 9-77
 SGR_CYAN, 9-45
 SGR_CYANBG, 9-55
 SGR_DEFAULT, 9-47
 SGR_DEFAULTBG, 9-57
 SGR_GREEN, 9-41
 SGR_GREENBG, 9-51
 SGR_ITALIC, 9-34
 SGR_MAGENTA, 9-44
 SGR_MAGENTABG, 9-54
 SGR_NEGATIVE, 9-36

```

```

 SGR_PRIMARY, 9-32
 SGR_RED, 9-40
 SGR_REDBG, 9-50
 SGR_UNDERSCORE, 9-35
 SGR_WHITE, 9-46
 SGR_WHITEBG, 9-56
 SGR_YELLOW, 9-42
 SGR_YELLOWBG, 9-52
 SHADE_BW, 27-1617
 SHADE_COLOR, 27-1619
 SHADE_GREYSCALE, 27-1618
 SHARED, 34-71, 40-65
 SHARED_LOCK, 16-41
 SHOWTITLE, 27-1398
 si_AltKeyMap, 27-655
 si_Buffer, 27-613
 si_BufferPos, 27-617
 si_CLeft, 27-630
 si_CTop, 27-631
 si_DispCount, 27-628
 si_DispPos, 27-620
 si_LayerPtr, 27-632
 si_LongInt, 27-645
 si_MaxChars, 27-618
 si_NumChars, 27-626
 si_SIZEOF, 27-657
 si_UndoBuffer, 27-615
 si_UndoPos, 27-625
 SIGBREAK, 16-150, 16-151, 16-152, 16-153
 SIGFLAC, 3-73
 SIMPLE_REFRESH, 27-1133
 SimpleSprite, 41-8
 SINGLE, 27-1589
 SIX_LPI, 27-1605
 SIZEBOTTOM, 27-1125
 SIZEBRIGHT, 27-1123
 SIZEOF_VIEW, 28-49
 sizes, 27-1257
 SIZEVERIF, 27-899
 SIZING, 27-483
 sm_ArgList, 42-32
 sm_ClipID, 8-62
 sm_Message, 42-27
 sm_Msg, 8-60
 sm_NumArgs, 42-30
 sm_Process, 42-28
 sm_Segment, 42-29
 sm_SIZEOF, 42-33
 sm_ToolWindow, 42-31
 sm_Unit, 8-61
 SMART_REFRESH, 27-1132
 some, 27-130, 27-227, 27-680, 27-1330
 something, 27-371
 sp_Msg, 18-107
 sp_Pkt, 18-108
 sp_SIZEOF, 18-109

```

SPECIAL\_ASPECT, 36-156  
 SPECIAL\_DENSITY1, 36-158  
 SPECIAL\_DENSITY2, 36-159  
 SPECIAL\_DENSITY3, 36-160  
 SPECIAL\_DENSITY4, 36-161  
 SPECIAL\_DENSITYMASK, 36-157  
 SPECIAL\_FRACCOLS, 36-154  
 SPECIAL\_FRACROWS, 36-155  
 SPECIAL\_FULLCOLS, 36-152  
 SPECIAL\_FULLROWS, 36-153  
 SPECIAL\_MILCOLS, 36-150  
 SPECIAL\_MILROWS, 36-151  
 SpecialInfo, 27-508, 27-608  
 specify, 27-1017  
 spr, 11-115  
 sprite, 27-1519  
 sprpt, 11-113  
 SRCA, 3-61  
 SRCB, 3-60  
 SRCC, 3-59  
 ss\_height, 41-10  
 ss\_num, 41-13  
 ss\_posctldata, 41-9  
 ss\_SIZEOF, 41-14  
 ss\_x, 41-11  
 ss\_y, 41-12  
 standard, 27-1042, 27-1215, 27-1361  
 StandardPacket, 18-106  
 still, 27-404  
 STOPPED, 37-68  
 strequ, 11-48  
 STRGADGET, 27-495  
 strhor, 11-50  
 String, 27-343, 27-614, 27-646, 34-68, 40-60  
 STRINGCENTER, 27-458  
 StringInfo, 27-609  
 STRINGRIGHT, 27-459  
 strings, 34-21, 40-21  
 strlong, 11-51  
 strvbl, 11-49  
 submitted, 27-635  
 SUD, 3-76  
 SUL, 3-77  
 SUPER\_BITMAP, 27-1134  
 SUPER\_UNUSED, 27-1175  
 SUPFRONT, 27-487  
 support, 27-1369  
 SUSERFLAGS, 20-13  
 SYSCADGET, 27-474  
 SYSREQUEST, 27-278  
 ta\_flags, 43-40  
 ta\_Name, 43-37  
 ta\_SIZEOF, 43-65, 43-41  
 ta\_Style, 43-39  
 ta\_YSize, 43-38  
 TALLDOT, 43-28

tasks, 18-15, 37-27, 48-27  
 TBC\_HCLRTAB, 9-90  
 TBC\_HCLRTABSALL, 9-91  
 TC\_SIZE, 18-34, 37-101  
 TD, 45-74  
 TD\_CHANGENUM, 45-81  
 TD\_CHANGESTATE, 45-82  
 TD\_FORMAT, 45-79, 45-96  
 TD\_LABELSIZE, 45-112  
 TD\_LASTCOMM, 45-85  
 TD\_MOTOR, 45-77, 45-94  
 TD\_NAME, 45-69  
 TD\_PROTSTATUS, 45-83, 45-85  
 TD\_REMOVE, 45-80  
 TD\_SECSHIFT, 45-50  
 TD\_SECTOR, 45-49  
 TD\_SEEK, 45-78, 45-95  
 TDERR\_BadDriveType, 45-133  
 TDERR\_BadHdrSum, 45-124  
 TDERR\_BadSecHdr, 45-127  
 TDERR\_BadSecID, 45-123  
 TDERR\_BadSecPreamble, 45-122  
 TDERR\_BadSecSum, 45-125  
 TDERR\_BadUnitNum, 45-132  
 TDERR\_DiskChanged, 45-129  
 TDERR\_DriveInUse, 45-134  
 TDERR\_NoMem, 45-131  
 TDERR\_NoSecHdr, 45-121  
 TDERR\_NotSpecified, 45-120  
 TDERR\_SeekError, 45-130  
 TDERR\_TooFewSecs, 45-126  
 TDERR\_WriteProt, 45-128  
 TDE\_EXTCOM, 45-92, 45-93, 45-94, 45-95, 45-96, 45-97, 45-98  
 TERMARRAY, 40-81  
 TERMARRAY\_0, 40-82  
 TERMARRAY\_1, 40-83  
 TERMARRAY\_SIZE, 40-84, 40-119  
 TextAttr, 43-36  
 TextFont, 43-45  
 tf\_Accessors, 43-54  
 tf\_Baseline, 43-51  
 tf\_BoldSmear, 43-52  
 tf\_CharData, 43-58  
 tf\_CharKern, 43-64  
 tf\_CharLoc, 43-61  
 tf\_CharSpace, 43-63  
 tf\_Flags, 43-49  
 tf\_HiChar, 43-57  
 tf\_LoChar, 43-56  
 tf\_Modulo, 43-60  
 tf\_SIZEOF, 43-56, 43-65  
 tf\_Style, 43-48  
 tf\_XSize, 43-50  
 tf\_YSize, 43-47  
 that, 27-115, 27-340, 27-790, 27-794, 27-826, 27-1257  
 TICKS\_PER\_SECOND, 16-52



TIMEREQUEST, 44-36  
 TIMERNAME, 44-26  
 TIMEVAL, 44-31  
 TmpRas, 38-15  
 TOGGLESELECT, 27-455  
   too, 27-743, 27-1424, 27-1426  
   top, 27-378  
   topaz, 27-1480  
 TOPAZ\_EIGHTY, 27-1496  
 TOPAZ\_SIXTY, 27-1497  
 TOPBORDER, 27-452  
   total, 27-561  
 TR\_ADDRREQUEST, 44-42  
 TR\_GETSYSTIME, 44-43  
   TR\_MakeBad, 46-12  
   TR\_NoMem, 46-11  
   TR\_NotUsed, 46-10  
   tr\_RasPtr, 38-16  
 TR\_SETSYSTIME, 44-44  
   tr\_Size, 38-17  
   tr\_SIZEOF, 38-18  
   trackdisk, 45-70  
   TV\_MICRO, 44-33  
   TV\_SECS, 44-32  
   TV\_SIZE, 25-142, 27-1512, 27-1513, 27-1514, 44-34, 44-37  
   two, 27-786  
 TXSCALE, 38-56  
   types, 12-21, 18-12, 27-1390, 32-6, 42-15, 48-15  
   typing, 27-472  
 ucl\_CopList, 10-49  
 ucl\_FirstCopList, 10-48  
   ucl\_Next, 10-47  
   ucl\_SIZEOF, 10-50  
   UCopList, 10-46  
 UNDERLINED, 43-22  
 UNIT\_MICROHZ, 44-23  
 UNIT\_VBLANK, 44-24  
 UnLoadSeq, 17-39  
 UnLock, 17-28  
   up, 27-337  
 UPKEYS, 19-31  
 US\_LEGAL, 27-1623  
 US\_LETTER, 27-1622  
   use, 27-80, 27-521, 27-786  
   used, 27-355, 27-809, 27-1075, 27-1222  
   V\_DUALPF, 47-13  
   v\_DxOffset, 47-50  
   v\_DyOffset, 47-49  
   V\_HAM, 47-16  
   V\_HRES, 47-14  
   V\_LACE, 47-15  
   v\_LOECprList, 47-47  
   v\_Modes, 47-51  
   V\_PFBA, 47-12  
   v\_SHECprList, 47-48  
   v\_SIZEOF, 47-52

V\_SPRITES, 47-17  
 v\_ViewPort, 47-46  
   value, 27-553  
   values, 27-1488  
 VANILLAKEY, 27-937  
   variable, 27-508, 27-608, 27-803, 27-1044, 27-1217,  
   27-1239, 27-1486  
   version, 27-19  
   vhposr, 11-22  
   vhposw, 11-42  
   view, 27-33, 28-24, 47-45  
   ViewPort, 10-36, 47-28  
   vp\_ClrIns, 47-33  
   vp\_ColorMap, 47-30  
   vp\_DHeight, 47-36  
   vp\_DspIns, 47-31  
   vp\_DWidth, 47-35  
   vp\_DxOffset, 47-37  
   vp\_DyOffset, 47-38  
   vp\_Modes, 47-39  
   vp\_Next, 47-29  
   vp\_RasInfo, 47-41  
   vp\_reserved, 47-40  
   vp\_SIZEOF, 27-1351, 47-42  
   vp\_SprIns, 47-32  
   vp\_UCopIns, 47-34  
   vposr, 11-21  
   VPOSRLOF, 14-40  
   vposw, 11-41  
   VS, 20-14, 20-15, 20-16, 20-17, 20-19, 20-20,  
   20-21, 20-22, 20-59  
 vs\_BorderLine, 20-88  
 vs\_ClearPath, 20-68  
 vs\_CollMask, 20-89  
   vs\_Depth, 20-82  
   vs\_DrawPath, 20-67  
   vs\_Height, 20-80  
   vs\_HitMask, 20-84  
   vs\_ImageData, 20-85  
   vs\_MeMask, 20-83  
 vs\_NextVSprite, 20-62  
   vs\_Oldx, 20-72  
   vs\_Oldy, 20-71  
   vs\_PlaneOnOff, 20-106  
   vs\_PlanePick, 20-105  
 vs\_PrevVSprite, 20-63  
   vs\_SIZEOF, 20-108  
   vs\_SprColors, 20-91  
   vs\_UserExt, 20-107  
   vs\_VSBob, 20-92  
   vs\_VSElags, 20-74  
   vs\_Width, 20-81  
   vs\_X, 20-79  
   vs\_Y, 20-78  
   vsize, 17-9  
 VSIZEBITS, 3-30

VSIZEMASK, 3-32  
 VSOVERFLOW, 20-22  
 VSPRITE, 20-14  
 W\_TRACTOR, 27-1625  
 wa\_Lock, 42-36  
 wa\_Name, 42-37  
 wa\_SIZEOF, 42-38  
 WaitForChar, 17-47  
 want, 27-1275, 27-1451  
 was, 27-276, 27-404  
 wasting, 27-794  
 way, 27-774  
 WB\_DISKMAGIC, 48-88  
 WB\_DISKVERSION, 48-89  
 WBArg, 42-35  
 WBDEVICE, 48-41  
 WBDISK, 48-36  
 WBDRAWER, 48-37  
 WBENCHCLOSE, 27-970  
 WBENCHMESSAGE, 27-930  
 WBENCHOPEN, 27-969  
 WBENCHSCREEN, 27-1395  
 WBENCHWINDOW, 27-1167  
 WBGARBAGE, 48-40  
 WBKICK, 48-42  
 WBOject, 48-98  
 WBPROJECT, 48-39  
 WBStartup, 42-26  
 WBT00L, 48-38  
 wd\_BlockPen, 27-1072  
 wd\_BorderBottom, 27-1036  
 wd\_BorderLeft, 27-1033  
 wd\_BorderRight, 27-1035  
 wd\_BorderRPort, 27-1037  
 wd\_BorderTop, 27-1034  
 wd\_CheckMark, 27-1079  
 wd\_Descendant, 27-1050  
 wd\_DetailPen, 27-1071  
 wd\_DMRequest, 27-1006  
 wd\_ExtData, 27-1100  
 wd\_FirstGadget, 27-1046  
 wd\_FirstRequest, 27-1004  
 wd\_Flags, 27-998  
 wd\_GZHeight, 27-1098  
 wd\_GZMouseX, 27-1092  
 wd\_GZMouseY, 27-1093  
 wd\_GZWidth, 27-1097  
 wd\_Height, 27-988  
 wd\_IDCMPFlags, 27-1062  
 wd\_LeftEdge, 27-985  
 wd\_MaxHeight, 27-996  
 wd\_MaxWidth, 27-995  
 wd\_MessageKey, 27-1000  
 wd\_MinHeight, 27-1069  
 wd\_MinWidth, 27-994

wd\_MouseX, 27-991  
 wd\_MouseY, 27-990  
 wd\_NextWindow, 27-983  
 wd\_Parent, 27-1049  
 wd\_Pointer, 27-1054  
 wd\_PtrHeight, 27-1055  
 wd\_PtrWidth, 27-1056  
 wd\_ReqCount, 27-1007  
 wd\_RPort, 27-1010  
 wd\_ScreenTitle, 27-1082  
 wd\_Size, 27-1106  
 wd\_Title, 27-1002  
 wd\_TopEdge, 27-986  
 wd\_UserData, 27-1103  
 wd\_UserPort, 27-1063  
 wd\_Width, 27-987  
 wd\_WindowPort, 27-1064  
 wd\_WLayer, 27-1104  
 wd\_WScreen, 27-1009  
 wd\_XOffset, 27-1057  
 wd\_YOffset, 27-1058  
 WDOWNBACK, 27-488  
 WDRAGGING, 27-484  
 WIDEDOT, 43-29  
 Window, 27-886, 27-981, 27-1008, 27-1040, 27-1138,  
 27-1205, 27-1275  
 WINDOWACTIVE, 27-1152  
 WINDOWCLOSE, 27-1116  
 WINDOWDEPTH, 27-1114  
 WINDOWDRAG, 27-1112  
 WINDOWREFRESH, 27-1166  
 Windows, 27-1251, 27-1424  
 WINDOWSIZING, 27-1110  
 WINDOWTICKED, 27-1168  
 wo, 48-128, 48-129, 48-130, 48-131  
 wo\_CurrentX, 48-117  
 wo\_CurrentY, 48-118  
 wo\_DefaultTool, 48-114  
 wo\_DrawerData, 48-115  
 wo\_Flags, 48-106  
 wo\_FreeList, 48-121  
 wo\_Gadget, 48-120  
 wo\_IconWin, 48-116  
 wo\_Lock, 48-124  
 wo\_MasterNode, 48-99  
 wo\_Name, 48-110  
 wo\_NameXOffset, 48-111  
 wo\_NameYOffset, 48-112  
 wo\_Parent, 48-103  
 wo\_SelectNode, 48-101  
 wo\_Siblings, 48-100  
 wo\_SIZEOF, 48-125  
 wo\_StackSize, 48-123  
 wo\_ToolTypes, 48-119  
 wo\_ToolWindow, 48-122  
 wo\_Type, 48-108

wo\_UseCount, 48-109  
wo\_UtilityNode, 48-102  
WRITE, 16-72, 17-21  
writing, 27-1031  
WROTEBREAK, 40-77  
WUPFRONT, 27-486  
XDISABLED, 40-63  
XOFFREAD, 40-74  
XOFFWRITE, 40-75  
you, 27-432, 27-440, 27-624, 27-649, 27-782,  
27-1017, 27-1025, 27-1730  
your, 27-782, 27-1040, 27-1251, 27-1447  
zero, 27-788

## Contents

devices/audio.i  
devices/bootblock.i  
devices/clipboard.i  
devices/console.i  
devices/gameport.i  
devices/input.i  
devices/inpotevent.i  
devices/keyboard.i  
devices/keymap.i  
devices/narrator.i  
devices/parallel.i  
devices/printer.i  
devices/prtbase.i  
devices/serial.i  
devices/timer.i  
devices/trackdisk.i

```
1 IFND DEVICES_AUDIO_I
2 DEVICES_AUDIO_I SET 1
3 *****
4 * Commodore-Amiga, Inc.
5 * audio.i
6 *****
7
8 IFND EXEC_IO_I
9 INCLUDE "exec/io.i"
10 ENDC
11
12 AUDIONAME MACRO
13 DC.B 'audio.device',0
14 ENDM
15
16 ADHARD_CHANNELS EQU 4
17
18 ADALLOC_MINPREC EQU -128
19 ADALLOC_MAXPREC EQU 127
20
21 ADCMD_FREE EQU CMD_NONSTD+0
22 ADCMD_SETPREC EQU CMD_NONSTD+1
23 ADCMD_FINISH EQU CMD_NONSTD+2
24 ADCMD_PERVOL EQU CMD_NONSTD+3
25 ADCMD_LOCK EQU CMD_NONSTD+4
26 ADCMD_WAITCYCLE EQU CMD_NONSTD+5
27 ADCMDB_NOUNIT EQU 5
28 ADCMDF_NOUNIT EQU 1<<5
29 ADCMD_ALLOCATE EQU ADCMDF_NOUNIT+0
30
31 ADIOB_PERVOL EQU 4
32 ADIOF_PERVOL EQU 1<<4
33 ADIOB_SYNCNCYCLE EQU 5
34 ADIOF_SYNCNCYCLE EQU 1<<5
35 ADIOB_NOWAIT EQU 6
36 ADIOF_NOWAIT EQU 1<<6
37 ADIOB_WRITEMESSAGE EQU 7
38 ADIOF_WRITEMESSAGE EQU 1<<7
39
40 ADIOERR_NOALLOCATION EQU -10
41 ADIOERR_ALLOCFAILED EQU -11
42 ADIOERR_CHANNELSTOLEN EQU -12
43
44 STRUCTURE IOAudio, IO_SIZE
45 WORD ioa_AllocKey
46 APTR ioa_Data
47 ULONG ioa_Length
48 UWORD ioa_Period
49 UWORD ioa_Volume
50 UWORD ioa_Cycles
51 STRUCT ioa_WriteMsg, MN_SIZE
52 LABEL ioa_SIZEEOF
53
54 ENDC
```

```

1 *****
2 * Commodore-Amiga, Inc. *
3 * bootblock.i *
4 *****
5 *****
6 *
7 * Source Control
8 * -----
9 *
10 * $Header: bootblock.i,v 27.1 85/06/24 13:15:16 neil Exp $
11 *
12 * $Locker: $
13 *
14 * $Log: bootblock.i,v $
15 * Revision 27.1 85/06/24 13:15:16 neil
16 * *** empty log message ***
17 *
18 * Revision 26.2 85/06/18 23:55:38 neil
19 * Added BBNAME definitions
20 *
21 * Revision 26.1 85/06/17 20:08:25 neil
22 * *** empty log message ***
23 *
24 *
25 *****
26
27 ***** BootBlock definition:
28
29 STRUCTURE BB,0
30 STRUCT BB_ID,4 * 4 character identifier
31 LONG BB_CHKSUM * boot block checksum (balance)
32 LONG BB_DOSBLOCK * reserved for DOS patch
33 LABEL BB_ENTRY * bootstrap entry point
34 LABEL BB_SIZE
35
36 BOOTSECTS equ 2 * 1K bootstrap
37
38 BBID_DOS macro * something that is bootable
39 dc.b 'DOS',0
40 endm
41
42 BBID_KICK macro * firmware image disk
43 dc.b 'KICK'
44 endm
45
46
47 BBNAME_DOS EQU (('D'<<24)!('O'<<16)!('S'<<8))
48 BBNAME_KICK EQU (('K'<<24)!('I'<<16)!('C'<<8)!('K'))

```

```

1 IFND DEVICES_CLIPBOARD_I
2 DEVICES_CLIPBOARD_I EQU 1
3 *****
4 * Commodore-Amiga, Inc. *
5 * clipboard.i *
6 *****
7 *****
8 *
9 * clipboard device command definitions
10 *
11 *****
12
13 IFND EXEC_NODES_I
14 INCLUDE "exec/nodes.i"
15 ENDC
16 IFND EXEC_LISTS_I
17 INCLUDE "exec/lists.i"
18 ENDC
19 IFND EXEC_PORTS_I
20 INCLUDE "exec/ports.i"
21 ENDC
22 IFND EXEC_IO_I
23 INCLUDE "exec/io.i"
24 ENDC
25
26 DEVINIT
27
28 DEVCMD CBD_POST
29 DEVCMD CBD_CURRENTREADID
30 DEVCMD CBD_CURRENTWRITEID
31
32 CBERR_OBSOLETEID EQU 1
33
34
35 STRUCTURE ClipboardUnitPartial,0
36 STRUCT cu_Node,LN_SIZE; ; list of units
37 ULONG cu_UnitNum; ; unit number for this unit
38 ; the remaining unit data is private to the device
39
40
41 STRUCTURE IOClipReq,0
42 STRUCT io_Message,MN_SIZE
43 APTR io_Device ; device node pointer
44 APTR io_Unit ; unit (driver private)
45 UWORD io_Command ; device command
46 BYTE io_Flags ; including QUICK and SATISFY
47 BYTE io_Error ; error or warning num
48 ULONG io_Actual ; number of bytes transferred
49 ULONG io_Length ; number of bytes requested
50 APTR io_Data ; either clip stream or post port
51 ULONG io_Offset ; offset in clip stream
52 LONG io_ClipID ; ordinal clip identifier
53 LABEL iocr_SIZEOF
54
55
56
57 PRIMARY_CLIP EQU 0 ; primary clip unit
58
59 STRUCTURE SatisfyMsg,0

```

```

60 STRUCT sm_Msg,MN_SIZE ; the length will be 6
61 UWORD sm_Unit ; which clip unit this is
62 LONG sm_ClipID ; the clip identifier of the post
63 LABEL satisfyMsg_SIZEOF
64
65 ENDC

```

D - 29

```

1 IFND DEVICES_CONSOLE_I
2 DEVICES_CONSOLE_I SET 1
3 *****
4 * Commodore-Amiga, Inc. *
5 * console.i *
6 *****
7 *****
8 *
9 * Console device command definitions
10 *
11 * Source Control
12 * -----
13 * $Header: console.i,v 1.4 85/11/13 15:13:21 kodiak Exp $
14 *
15 * $Locker: $
16 *
17 *****
18
19 IFND EXEC_IO_I
20 INCLUDE "exec/io.i"
21 ENDC
22
23 ***** Console commands *****
24 DEVINIT
25
26 DEVCMD CD_ASKKEYMAP
27 DEVCMD CD_SETKEYMAP
28
29
30 ***** SGR parameters
31
32 SGR_PRIMARY EQU 0
33 SGR_BOLD EQU 1
34 SGR_ITALIC EQU 3
35 SGR_UNDERSCORE EQU 4
36 SGR_NEGATIVE EQU 7
37
38 * these names refer to the ANSI standard, not the implementation
39 SGR_BLACK EQU 30
40 SGR_RED EQU 31
41 SGR_GREEN EQU 32
42 SGR_YELLOW EQU 33
43 SGR_BLUE EQU 34
44 SGR_MAGENTA EQU 35
45 SGR_CYAN EQU 36
46 SGR_WHITE EQU 37
47 SGR_DEFAULT EQU 39
48
49 SGR_BLACKBG EQU 40
50 SGR_REDBG EQU 41
51 SGR_GREENBG EQU 42
52 SGR_YELLOWBG EQU 43
53 SGR_BLUEBG EQU 44
54 SGR_MAGENTABG EQU 45
55 SGR_CYANBG EQU 46
56 SGR_WHITEBG EQU 47
57 SGR_DEFAULTBG EQU 49
58
59 * these names refer to the implementation, they are the preferred

```

60 \* names for use with the Amiga console device.

```

61 SGR_CLR0 EQU 30
62 SGR_CLR1 EQU 31
63 SGR_CLR2 EQU 32
64 SGR_CLR3 EQU 33
65 SGR_CLR4 EQU 34
66 SGR_CLR5 EQU 35
67 SGR_CLR6 EQU 36
68 SGR_CLR7 EQU 37
69
70 SGR_CLR0BG EQU 40
71 SGR_CLR1BG EQU 41
72 SGR_CLR2BG EQU 42
73 SGR_CLR3BG EQU 43
74 SGR_CLR4BG EQU 44
75 SGR_CLR5BG EQU 45
76 SGR_CLR6BG EQU 46
77 SGR_CLR7BG EQU 47
78
79
80 ***** DSR parameters
81
82 DSR_CPR EQU 6
83
84 ***** CTC parameters
85 CTC_HSETTAB EQU 0
86 CTC_HCLRTAB EQU 2
87 CTC_HCLRTABSALL EQU 5
88
89 ***** TBC parameters
90 TBC_HCLRTAB EQU 0
91 TBC_HCLRTABSALL EQU 3
92
93 ***** SM and RM parameters
94 M_LNM EQU 20 ; linefeed newline mode
95 M_ASM MACRO
96 DC.B '>1' ; auto scroll mode
97 ENDM
98 M_AWM MACRO
99 DC.B '?7' ; auto wrap mode
100 ENDM
101
102 ENDC

```

```

1 IFND DEVICES_GAMEPORT_I
2 DEVICES_GAMEPORT_I SET 1
3 *****
4 * Commodore-Amiga, Inc. *
5 * gameport.i *
6 *****
7 *****
8 *
9 * Game Port device command definitions
10 *
11 *****
12
13 IFND EXEC_IO_I
14 INCLUDE "exec/io.i"
15 ENDC
16
17
18 ***** GamePort commands *****
19 DEVINIT
20
21 DEVCMD GPD_READEVENT
22 DEVCMD GPD_ASKCTYPE
23 DEVCMD GPD_SETCTYPE
24 DEVCMD GPD_ASKTRIGGER
25 DEVCMD GPD_SETTRIGGER
26
27 ***** GamePort structures *****
28
29 * gpt_Keys
30 BITDEF GPT,DOWNKEYS,0
31 BITDEF GPT,UPKEYS,1
32
33 STRUCTURE GamePortTrigger,0
34 UWORD gpt_Keys ;key transition triggers
35 UWORD gpt_Timeout ;time trigger (vertical blank units)
36 UWORD gpt_XDelta ;X distance trigger
37 UWORD gpt_YDelta ;Y distance trigger
38 LABEL gpt_SIZEOF
39
40 ***** Controller Types *****
41 GPCT_ALLOCATED EQU -1 ; allocated by another user
42 GPCT_NOCONTROLLER EQU 0
43
44 GPCT_MOUSE EQU 1
45 GPCT_RELJOYSTICK EQU 2
46 GPCT_ABSJOYSTICK EQU 3
47
48
49 ***** Errors *****
50 GPDERR_SETCTYPE EQU 1 ; this controller not valid at this time
51
52 ENDC

```

```

1 IFND DEVICES_INPUT_I
2 DEVICES_INPUT_I SET 1
3 *****
4 * Commodore-Amiga, Inc. *
5 * input.i *
6 *****
7 *****
8 *
9 * input device command definitions
10 *
11 *****
12
13 IFND EXEC_IO_I
14 INCLUDE "exec/io.i"
15 ENDC
16
17 DEVINIT
18
19 DEVCMD IND_ADDHANDLER
20 DEVCMD IND_REMHANDLER
21 DEVCMD IND_WRITEEVENT
22 DEVCMD IND_SETTHRESH
23 DEVCMD IND_SETPERIOD
24 DEVCMD IND_SETMPORT
25 DEVCMD IND_SETMTYPE
26 DEVCMD IND_SETMTRIG
27
28 ENDC

```

```

1 IFND DEVICES_INPUTEVENT_I
2 DEVICES_INPUTEVENT_I SET 1
3 *****
4 * Commodore-Amiga, Inc. *
5 * inputevent.i *
6 *****
7 *****
8 *
9 * input event definitions
10 *
11 *****
12
13 IFND DEVICES_TIMER_I
14 INCLUDE "devices/timer.i"
15 ENDC
16
17 *----- constants -----
18
19 * --- InputEvent.ie_Class ---
20 * A NOP input event
21 IECLASS_NULL EQU $00
22 * A raw keycode from the keyboard device
23 IECLASS_RAWKEY EQU $01
24 * A raw mouse report from the game port device
25 IECLASS_RAWMOUSE EQU $02
26 * A private console event
27 IECLASS_EVENT EQU $03
28 * A Pointer Position report
29 IECLASS_POINTERPOS EQU $04
30 * A timer event
31 IECLASS_TIMER EQU $06
32 * select button pressed down over a Gadget (address in ie_EventAddress)
33 IECLASS_GADGETDOWN EQU $07
34 * select button released over the same Gadget (address in ie_EventAddress)
35 IECLASS_GADGETUP EQU $08
36 * some Requester activity has taken place. See Codes REQCLEAR and REQSET
37 IECLASS_REQUESTER EQU $09
38 * this is a Menu Number transmission (Menu number is in ie_Code)
39 IECLASS_MENULIST EQU $0A
40 * User has selected the active Window's Close Gadget
41 IECLASS_CLOSEWINDOW EQU $0B
42 * this Window has a new size
43 IECLASS_SIZEWINDOW EQU $0C
44 * the Window pointed to by ie_EventAddress needs to be refreshed
45 IECLASS_REFRESHWINDOW EQU $0D
46 * new preferences are available
47 IECLASS_NEWPREFS EQU $0E
48 * the disk has been removed
49 IECLASS_DISKREMOVED EQU $0F
50 * the disk has been inserted
51 IECLASS_DISKINSERTED EQU $10
52 * the window is about to be been made active
53 IECLASS_ACTIVEWINDOW EQU $11
54 * the window is about to be made inactive
55 IECLASS_INACTIVEWINDOW EQU $12
56
57 * the last class
58 IECLASS_MAX EQU $12
59

```



```

60 * --- InputEvent.ie_Code ---
61 * IECLASS_RAWKEY
62 IECODE_UP_PREFIX EQU $80
63 IECODEB_UP_PREFIX EQU 7
64 IECODE_KEY_CODE_FIRST EQU $00
65 IECODE_KEY_CODE_LAST EQU $77
66 IECODE_COMM_CODE_FIRST EQU $78
67 IECODE_COMM_CODE_LAST EQU $7F
68
69 * IECLASS_ANSI
70 IECODE_C0_FIRST EQU $00
71 IECODE_C0_LAST EQU $1F
72 IECODE_ASCII_FIRST EQU $20
73 IECODE_ASCII_LAST EQU $7E
74 IECODE_ASCII_DEL EQU $7F
75 IECODE_C1_FIRST EQU $80
76 IECODE_C1_LAST EQU $9F
77 IECODE_LATIN1_FIRST EQU $A0
78 IECODE_LATIN1_LAST EQU $FF
79
80 * IECLASS_RAWMOUSE
81 IECODE_LBUTTON EQU $68 ; also uses IECODE_UP_PREFIX
82 IECODE_RBUTTON EQU $69 ;
83 IECODE_MBUTTON EQU $6A ;
84 IECODE_NOBUTTON EQU $FF
85
86 * IECLASS_EVENT
87 IECODE_NEWACTIVE EQU $01 ; active input window changed
88
89 * IECLASS_REQUESTER Codes
90 * REQSET is broadcast when the first Requester (not subsequent ones) opens
91 * in the Window
92 IECODE_REQSET EQU $01
93 * REQCLEAR is broadcast when the last Requester clears out of the Window
94 IECODE_REQCLEAR EQU $00
95
96
97 * --- InputEvent.ie_Qualifier ---
98 IEQUALIFIER_LSHIFT EQU $0001
99 IEQUALIFIERB_LSHIFT EQU 0
100 IEQUALIFIER_RSHIFT EQU $0002
101 IEQUALIFIERB_RSHIFT EQU 1
102 IEQUALIFIER_CAPSLOCK EQU $0004
103 IEQUALIFIERB_CAPSLOCK EQU 2
104 IEQUALIFIER_CONTROL EQU $0008
105 IEQUALIFIERB_CONTROL EQU 3
106 IEQUALIFIER_LALT EQU $0010
107 IEQUALIFIERB_LALT EQU 4
108 IEQUALIFIER_RALT EQU $0020
109 IEQUALIFIERB_RALT EQU 5
110 IEQUALIFIER_LCOMMAND EQU $0040
111 IEQUALIFIERB_LCOMMAND EQU 6
112 IEQUALIFIER_RCOMMAND EQU $0080
113 IEQUALIFIERB_RCOMMAND EQU 7
114 IEQUALIFIER_NUMERICPAD EQU $0100
115 IEQUALIFIERB_NUMERICPAD EQU 8
116 IEQUALIFIER_REPEAT EQU $0200
117 IEQUALIFIERB_REPEAT EQU 9
118 IEQUALIFIER_INTERRUPT EQU $0400
119 IEQUALIFIERB_INTERRUPT EQU 10
120 IEQUALIFIER_MULTIBROADCAST EQU $0800
121 IEQUALIFIERB_MULTIBROADCAST EQU 11
122 IEQUALIFIER_LBUTTON EQU $1000
123 IEQUALIFIERB_LBUTTON EQU 12
124 IEQUALIFIER_RBUTTON EQU $2000
125 IEQUALIFIERB_RBUTTON EQU 13
126 IEQUALIFIER_MBUTTON EQU $4000
127 IEQUALIFIERB_MBUTTON EQU 14
128 IEQUALIFIER_RELATIVEMOUSE EQU $8000
129 IEQUALIFIERB_RELATIVEMOUSE EQU 15
130
131 *----- InputEvent -----
132
133 STRUCTURE InputEvent,0
134 APTR ie_NextEvent ; the chronologically next event
135 UBYTE ie_Class ; the input event class
136 UBYTE ie_SubClass ; optional subclass of the class
137 UWORD ie_Code ; the input event code
138 UWORD ie_Qualifier ; qualifiers in effect for the event
139 LABEL ie_EventAddress ; a pointer parameter for an event
140 WORD ie_X ; the pointer position for the event,
141 WORD ie_Y ; usually in canvas relative coords
142 STRUCT ie_TimeStamp,TV_SIZE ; the system tick at the event
143 LABEL ie_SIZEOF
144
145 ENDC

```

```

1 IFND DEVICES_KEYBOARD_I
2 DEVICES_KEYBOARD_I SET 1
3 *****
4 * Commodore-Amiga, Inc. *
5 * keyboard.i *
6 *****
7 *****
8 *
9 * Keyboard device command definitions
10 *
11 *****
12
13 IFND EXEC_IO_I
14 INCLUDE "exec/io.i"
15 ENDC
16
17 DEVINIT
18
19 DEVCMD KBD_READEVENT
20 DEVCMD KBD_READMATRIX
21 DEVCMD KBD_ADDRESETHANDLER
22 DEVCMD KBD_REMRESETHANDLER
23 DEVCMD KBD_RESETHANDLERDONE
24
25 ENDC

```

```

1 IFND DEVICES_KEYMAP_I
2 DEVICES_KEYMAP_I SET 1
3 *****
4 * Commodore-Amiga, Inc. *
5 * keymap.i *
6 *****
7 *****
8 *
9 * console.device key map definitions
10 *
11 *****
12
13 STRUCTURE KeyMap,0
14 APTR km_LoKeyMapTypes
15 APTR km_LoKeyMap
16 APTR km_LoCapsable
17 APTR km_LoRepeatable
18 APTR km_HiKeyMapTypes
19 APTR km_HiKeyMap
20 APTR km_HiCapsable
21 APTR km_HiRepeatable
22 LABEL km_SIZEOF
23
24
25 KCB_NOP EQU 7
26 KCF_NOP EQU $80
27
28 KC_NOQUAL EQU 0
29 KC_VANILLA EQU 7
30 KCF_SHIFT EQU $01
31 KCF_ALT EQU $02
32 KCB_CONTROL EQU 2
33 KCF_CONTROL EQU $04
34 KCB_DOWNUP EQU 3
35 KCF_DOWNUP EQU $08
36
37 KCB_STRING EQU 6
38 KCF_STRING EQU $40
39
40 ENDC

```

; note that SHIFT+ALT+CTRL is VANILLA

```

1 IFND DEVICES_NARRATOR_I
2 DEVICES_NARRATOR_I SET 1
3 *****
4 * Commodore-Amiga, Inc. *
5 * narrator.i *
6 *****
7
8 IFND EXEC_IO_I
9 INCLUDE "exec/io.i"
10 ENDC
11
12 *----- DEFAULT VALUES, USER PARMS, AND GENERAL CONSTANTS
13
14 DEFPITCH EQU 110 ;DEFAULT PITCH
15 DEFPRATE EQU 150 ;DEFAULT RATE
16 DEFVOL EQU 64 ;DEFAULT VOLUME (FULL)
17 DEFREQ EQU 22200 ;DEFAULT SAMPLING FREQUENCY
18 NATURALFO EQU 0 ;NATURAL FO CONTOURS
19 ROBOTICFO EQU 1 ;MONOTONE FO
20 MALE EQU 0 ;MALE SPEAKER
21 FEMALE EQU 1 ;FEMALE SPEAKER
22 DEFSEX EQU MALE ;DEFAULT SEX
23 DEFMODE EQU NATURALFO ;DEFAULT MODE
24
25 * Parameter bounds
26
27 MINRATE EQU 40 ;MINIMUM SPEAKING RATE
28 MAXRATE EQU 400 ;MAXIMUM SPEAKING RATE
29 MINPITCH EQU 65 ;MINIMUM PITCH
30 MAXPITCH EQU 320 ;MAXIMUM PITCH
31 MINFREQ EQU 5000 ;MINIMUM SAMPLING FREQUENCY
32 MAXFREQ EQU 28000 ;MAXIMUM SAMPLING FREQUENCY
33 MINVOL EQU 0 ;MINIMUM VOLUME
34 MAXVOL EQU 64 ;MAXIMUM VOLUME
35
36 * Driver error codes
37
38 ND_NotUsed EQU -1 ;
39 ND_NoMem EQU -2 ;Can't allocate memory
40 ND_NoAudLib EQU -3 ;Can't open audio device
41 ND_MakeBad EQU -4 ;Error in MakeLibrary call
42 ND_UnitErr EQU -5 ;Unit other than 0
43 ND_CantAlloc EQU -6 ;Can't allocate the audio channel
44 ND_Unimpl EQU -7 ;Unimplemented command
45 ND_NoWrite EQU -8 ;Read for mouth shape without write
46 ND_Expunged EQU -9 ;Can't open, deferred expunge bit set
47 ND_PhonErr EQU -20 ;Phoneme code spelling error
48 ND_RateErr EQU -21 ;Rate out of bounds
49 ND_PitchErr EQU -22 ;Pitch out of bounds
50 ND_SexErr EQU -23 ;Sex not valid
51 ND_ModeErr EQU -24 ;Mode not valid
52 ND_FreqErr EQU -25 ;Sampling freq out of bounds
53 ND_VolErr EQU -26 ;Volume out of bounds
54
55
56
57 * ;----- Write IORequest block
58 STRUCTURE NDI,IOSTD_SIZE
59 UWORD NDI_RATE ;Speaking rate in words/minute

```

```

60 UWORD NDI_PITCH ;Baseline pitch in Hertz
61 UWORD NDI_MODE ;F0 mode
62 UWORD NDI_SEX ;Speaker sex
63 APTR NDI_CHMASKS ;Pointer to audio channel masks
64 UWORD NDI_NUMMASKS ;Size of channel masks array
65 UWORD NDI_VOLUME ;Channel volume
66 UWORD NDI_SAMPFREQ ;Sampling frequency
67 UBYTE NDI_MOUTHS ;Generate mouths? (Boolean value)
68 UBYTE NDI_CHANMASK ;Actual channel mask used (internal use)
69 UBYTE NDI_NUMCHAN ;Number of channels used (internal use)
70 UBYTE NDI_PAD ;For alignment
71 LABEL NDI_SIZE ;Size of Narrator IORequest block
72
73
74 * ;----- Mouth read IORB
75 STRUCTURE MRB,NDI_SIZE
76 UBYTE MRB_WIDTH ;Mouth width
77 UBYTE MRB_HEIGHT ;Mouth height
78 UBYTE MRB_SHAPE ;Compressed shape (height/width)
79 UBYTE MRB_PAD ;Alignment
80 LABEL MRB_SIZE
81
82
83 ENDC
84

```

D - 34

```

1 *****
2 * Commodore-Amiga, Inc. *
3 * parallel.i *
4 *****
5 *****
6 *
7 * external declarations for Parallel Port Driver
8 *
9 * SOURCE CONTROL
10 *
11 * $Header: parallel.i,v 25.0 85/03/27 19:14:15 tomp Exp $
12 *
13 * $Locker: $
14 *
15 *****
16
17 IFND DEVICES_PARALLEL_I
18 DEVICES_PARALLEL_I SET 1
19
20 IFND EXEC_STRINGS_I
21 include 'exec/strings.i'
22
23
24 IFND EXEC_IO_I
25 include 'exec/io.i'
26
27
28 *-----
29 *
30 * Driver error definitions
31 *
32 *-----
33
34 ParErr_DevBusy EQU 1
35 ParErr_BufTooBig EQU 2
36 ParErr_InvParam EQU 3
37 ParErr_LineErr EQU 4
38 ParErr_NotOpen EQU 5
39 ParErr_PortReset EQU 6
40 ParErr_InitErr EQU 7
41
42 *-----
43 *
44 * Useful constants
45 *
46 *-----
47 *
48 PDCMD_QUERY EQU CMD_NONSTD
49 PDCMD_SETPARAMS EQU CMD_NONSTD+1
50 Par_DEVPFINISH EQU 10 ; number of device comands
51 *
52 *-----
53 *
54 * Driver Specific Commands
55 *
56 *-----
57 *
58 *-- PARALLELNAME is a generic macro to get the name of the driver. This
59 *-- way if the name is ever changed you will pick up the change

```

```

60 *-- automatically.
61 *--
62 *-- Normal usage would be:
63 *--
64 *-- internalName: PARALLELNAME
65 *--
66
67 PARALLELNAME: MACRO
68 STRING 'parallel.device'
69 ENDM
70
71 BITDEF PAR,SHARED,5 ; PARFLAGS non-exclusive access
72 BITDEF PAR,RAD_BOOGIE,3 ; " (not yet implemented)
73 BITDEF PAR,EOFMODE,1 ; " EOF mode enabled bit
74 BITDEF IOPAR,QUEUED,6 ; IO_FLAGS rqst-queued bit
75 BITDEF IOPAR,ABORT,5 ; " rqst-aborted bit
76 BITDEF IOPAR,ACTIVE,4 ; " rqst-qed-or-current bit
77 BITDEF IOPT,RWDIR,3 ; IO_STATUS read=0,write=1
78 BITDEF IOPT,PBUSY,2 ; " printer in busy toggle
79 BITDEF IOPT,PAPEROUT,1 ; " paper out
80 BITDEF IOPT,PSEL,0 ; " printer selected
81 *
82 *
83 *****
84
85 STRUCTURE PTERMARRAY,0
86 ULONG PTERMARRAY_0
87 ULONG PTERMARRAY_1
88 LABEL PTERMARRAY_SIZE
89
90 *****
91 * CAUTION !!! IF YOU ACCESS the parallel.device, you MUST (!!!!) use an
92 * IOEXTPAR-sized structure or you may overlay innocent memory, okay ?!
93 *****
94
95 STRUCTURE IOEXTPAR,IOSTD_SIZE
96
97 * STRUCT MsgNode
98 * 0 APTR Succ
99 * 4 APTR Pred
100 * 8 UBYTE Type
101 * 9 UBYTE Pri
102 * A APTR Name
103 * E APTR ReplyPort
104 * 12 UWORD MNLength
105 * STRUCT IOExt
106 * 14 APTR IO_DEVICE
107 * 18 APTR IO_UNIT
108 * 1C UWORD IO_COMMAND
109 * 1E UBYTE IO_FLAGS
110 * 1F UBYTE IO_ERROR
111 * STRUCT IOStdExt
112 * 20 ULONG IO_ACTUAL
113 * 24 ULONG IO_LENGTH
114 * 28 APTR IO_DATA
115 * 2C ULONG IO_OFFSET
116 *
117 *
118 * 30
119 ULONG IO_PEXTFLAGS ; (not used) flag extension area

```

```

120 UBYTE IO_PARSTATUS ; device status (see bit defs above)
121 UBYTE IO_PARFLAGS ; see PARFLAGS bit definitions above
122 STRUCT IO_PTERMARRAY, PTERMARRAY_SIZE ; termination char array
123 LABEL IOEXTPar_SIZE
124
125
126

```

```

1 IFND DEVICES_PRINTER_I
2 DEVICES_PRINTER_I EQU 1
3 *****
4 * Commodore-Amiga, Inc. *
5 * printer.i *
6 *****
7 *****
8 *
9 * printer device command definitions
10 *
11 * Source Control
12 * -----
13 * $Header: printer.i,v 1.2 85/10/09 16:16:27 kodiak Exp $
14 *
15 * $Locker: $
16 *
17 *****
18
19 IFND EXEC_NODES_I
20 INCLUDE "exec/nodes.i"
21 ENDC
22
23 IFND EXEC_LISTS_I
24 INCLUDE "exec/lists.i"
25 ENDC
26
27 IFND EXEC_PORTS_I
28 INCLUDE "exec/ports.i"
29 ENDC
30
31 IFND EXEC_IO_I
32 INCLUDE "exec/io.i"
33 ENDC
34
35 DEVINIT
36
37 DEVCMD PRD_RAWWRITE
38 DEVCMD PRD_PRTCOMMAND
39 DEVCMD PRD_DUMPREPORT
40
41 ;***** printer definitions
42 aRIS EQU 0 ; ESCc reset ISO
43 aRIN EQU 1 ; ESC#1 initialize +++
44 aIND EQU 2 ; ESCD lf ISO
45 aNEL EQU 3 ; ESCE return,lf ISO
46 aRI EQU 4 ; ESCM reverse lf ISO
47
48 aSGR0 EQU 5 ; ESC[0m normal char set ISO
49 aSGR3 EQU 6 ; ESC[3m italics on ISO
50 aSGR23 EQU 7 ; ESC[23m italics off ISO
51 aSGR4 EQU 8 ; ESC[4m underline on ISO
52 aSGR24 EQU 9 ; ESC[24m underline off ISO
53 aSGR1 EQU 10 ; ESC[1m boldface on ISO
54 aSGR22 EQU 11 ; ESC[22m boldface off ISO
55 aSFC EQU 12 ; SGR30-39 set foreground color ISO
56 aSBC EQU 13 ; SGR40-49 set background color ISO
57
58 aSHORP0 EQU 14 ; ESC[0w normal pitch DEC
59 aSHORP2 EQU 15 ; ESC[2w elite on DEC

```

|     |         |     |                                      |               |     |                       |                     |                                         |     |
|-----|---------|-----|--------------------------------------|---------------|-----|-----------------------|---------------------|-----------------------------------------|-----|
| 60  | aSHORP1 | EQU | 16 ; ESC[1w elite off                | DEC           | 120 | aTBC0                 | EQU                 | 69 ; ESC[0g Clr horiz tab               | ISO |
| 61  | aSHORP4 | EQU | 17 ; ESC[4w condensed fine on        | DEC           | 121 | aTBC3                 | EQU                 | 70 ; ESC[3g Clear all h tab             | ISO |
| 62  | aSHORP3 | EQU | 18 ; ESC[3w condensed off            | DEC           | 122 | aTBC1                 | EQU                 | 71 ; ESC[1g Clr vertical tabs           | ISO |
| 63  | aSHORP6 | EQU | 19 ; ESC[6w enlarged on              | DEC           | 123 | aTBC4                 | EQU                 | 72 ; ESC[4g Clr all v tabs              | ISO |
| 64  | aSHORP5 | EQU | 20 ; ESC[5w enlarged off             | DEC           | 124 | aTBCALL               | EQU                 | 73 ; ESC#4 Clr all h & v tabs           | +++ |
| 65  |         |     |                                      |               | 125 | aTBSALL               | EQU                 | 74 ; ESC#5 Set default tabs             | +++ |
| 66  | aDEN6   | EQU | 21 ; ESC[6"z shadow print on         | DEC (sort of) | 126 | aEXTEND               | EQU                 | 75 ; ESC[Pn"x extended commands         | +++ |
| 67  | aDEN5   | EQU | 22 ; ESC[5"z shadow print off        | DEC           | 127 |                       |                     |                                         |     |
| 68  | aDEN4   | EQU | 23 ; ESC[4"z doublestrike on         | DEC           | 128 |                       |                     |                                         |     |
| 69  | aDEN3   | EQU | 24 ; ESC[3"z doublestrike off        | DEC           | 129 | STRUCTURE             | IOPrCmdReq, IO_SIZE |                                         |     |
| 70  | aDEN2   | EQU | 25 ; ESC[2"z NLQ on                  | DEC           | 130 | UWORD                 | io_PrtCommand       | ; printer command                       |     |
| 71  | aDEN1   | EQU | 26 ; ESC[1"z NLQ off                 | DEC           | 131 | UBYTE                 | io_Parm0            | ; first command parameter               |     |
| 72  |         |     |                                      |               | 132 | UBYTE                 | io_Parm1            | ; second command parameter              |     |
| 73  | aSUS2   | EQU | 27 ; ESC[2v superscript on           | +++           | 133 | UBYTE                 | io_Parm2            | ; third command parameter               |     |
| 74  | aSUS1   | EQU | 28 ; ESC[1v superscript off          | +++           | 134 | UBYTE                 | io_Parm3            | ; fourth command parameter              |     |
| 75  | aSUS4   | EQU | 29 ; ESC[4v subscript on             | +++           | 135 | LABEL                 | iopecr_SIZEEOF      |                                         |     |
| 76  | aSUS3   | EQU | 30 ; ESC[3v subscript off            | +++           | 136 |                       |                     |                                         |     |
| 77  | aSUS0   | EQU | 31 ; ESC[0v normalize the line       | +++           | 137 | STRUCTURE             | IODRPreq, IO_SIZE   |                                         |     |
| 78  | aPLU    | EQU | 32 ; ESC[L partial line up           | ISO           | 138 | APTR                  | io_RastPort         | ; raster port                           |     |
| 79  | aPLD    | EQU | 33 ; ESC[K partial line down         | ISO           | 139 | APTR                  | io_ColorMap         | ; color map                             |     |
| 80  |         |     |                                      |               | 140 | ULONG                 | io_Modes            | ; graphics viewport modes               |     |
| 81  | aFNT0   | EQU | 34 ; ESC(B US char set               | DEC           | 141 | UWORD                 | io_SrcX             | ; source x origin                       |     |
| 82  | aFNT1   | EQU | 35 ; ESC(R French char set           | DEC           | 142 | UWORD                 | io_SrcY             | ; source y origin                       |     |
| 83  | aFNT2   | EQU | 36 ; ESC(K German char set           | DEC           | 143 | UWORD                 | io_SrcWidth         | ; source x width                        |     |
| 84  | aFNT3   | EQU | 37 ; ESC(A UK char set               | DEC           | 144 | UWORD                 | io_SrcHeight        | ; source x height                       |     |
| 85  | aFNT4   | EQU | 38 ; ESC(E Danish I char set         | DEC           | 145 | LONG                  | io_DestCols         | ; destination x width                   |     |
| 86  | aFNT5   | EQU | 39 ; ESC(H Sweden char set           | DEC           | 146 | LONG                  | io_DestRows         | ; destination y height                  |     |
| 87  | aFNT6   | EQU | 40 ; ESC(Y Italian char set          | DEC           | 147 | UWORD                 | io_Special          | ; option flags                          |     |
| 88  | aFNT7   | EQU | 41 ; ESC(Z Spanish char set          | DEC           | 148 | LABEL                 | iodrpr_SIZEEOF      |                                         |     |
| 89  | aFNT8   | EQU | 42 ; ESC(J Japanese char set         | +++           | 149 |                       |                     |                                         |     |
| 90  | aFNT9   | EQU | 43 ; ESC(6 Norweign char set         | DEC           | 150 | SPECIAL_MILCOLS       | EQU                 | \$01 ; DestCols specified in 1/1000"    |     |
| 91  | aFNT10  | EQU | 44 ; ESC(C Danish II char set        | +++           | 151 | SPECIAL_MILROWS       | EQU                 | \$02 ; DestRows specified in 1/1000"    |     |
| 92  |         |     |                                      |               | 152 | SPECIAL_FULLCOLS      | EQU                 | \$04 ; make DestCols maximum possible   |     |
| 93  | aPROP2  | EQU | 45 ; ESC[2p proportional on          | +++           | 153 | SPECIAL_FULLROWS      | EQU                 | \$08 ; make DestRows maximum possible   |     |
| 94  | aPROP1  | EQU | 46 ; ESC[1p proportional off         | +++           | 154 | SPECIAL_FRACCOLS      | EQU                 | \$10 ; DestCols is fraction of FULLCOLS |     |
| 95  | aPROPO  | EQU | 47 ; ESC[0p proportional clear       | +++           | 155 | SPECIAL_FRACROWS      | EQU                 | \$20 ; DestRows is fraction of FULLROWS |     |
| 96  | aTSS    | EQU | 48 ; ESC[n E set proportional offset | ISO           | 156 | SPECIAL_ASPECT        | EQU                 | \$80 ; ensure correct aspect ratio      |     |
| 97  | aJFY5   | EQU | 49 ; ESC[5 F auto left justify       | ISO           | 157 | SPECIAL_DENSITYMASK   | EQU                 | \$F00 ; masks out density bits          |     |
| 98  | aJFY7   | EQU | 50 ; ESC[7 F auto right justiy       | ISO           | 158 | SPECIAL_DENSITY1      | EQU                 | \$100 ; lowest res                      |     |
| 99  | aJFY6   | EQU | 51 ; ESC[6 F auto full justify       | ISO           | 159 | SPECIAL_DENSITY2      | EQU                 | \$200 ; next res                        |     |
| 100 | aJFY0   | EQU | 52 ; ESC[0 F auto justify off        | ISO           | 160 | SPECIAL_DENSITY3      | EQU                 | \$300 ; next res                        |     |
| 101 | aJFY2   | EQU | 53 ; ESC[2 F word space(auto center) | ISO (special) | 161 | SPECIAL_DENSITY4      | EQU                 | \$400 ; highest res                     |     |
| 102 | aJFY3   | EQU | 54 ; ESC[3 F letter space (justify)  | ISO (special) | 162 |                       |                     |                                         |     |
| 103 |         |     |                                      |               | 163 | PDERR_CANCEL          | EQU                 | 1 ; user canceled a printer timeout     |     |
| 104 | aVERP0  | EQU | 55 ; ESC[0z 1/8" line spacing        | +++           | 164 | PDERR_NOTGRAPHICS     | EQU                 | 2 ; printer cannot output graphics      |     |
| 105 | aVERP1  | EQU | 56 ; ESC[1z 1/6" line spacing        | +++           | 165 | PDERR_INVERTHAM       | EQU                 | 3 ; cannot invert hold & modify print   |     |
| 106 | aSLPP   | EQU | 57 ; ESC[nt set form length n        | DEC           | 166 | PDERR_BADDIMENSION    | EQU                 | 4 ; print dimensions illegal            |     |
| 107 | aPERF   | EQU | 58 ; ESC[nq perf skip n (n>0)        | +++           | 167 | PDERR_DIMENSIONOVFLOW | EQU                 | 5 ; print dimensions too large          |     |
| 108 | aPERFO  | EQU | 59 ; ESC[0q perf skip off            | +++           | 168 | PDERR_INTERNALMEMORY  | EQU                 | 6 ; no memory for internal variables    |     |
| 109 |         |     |                                      |               | 169 | PDERR_BUFFERMEMORY    | EQU                 | 7 ; no memory for print buffer          |     |
| 110 | aLMS    | EQU | 60 ; ESC#9 Left margin set           | +++           | 170 |                       |                     |                                         |     |
| 111 | aRMS    | EQU | 61 ; ESC#0 Right margin set          | +++           | 171 | ENDC                  |                     |                                         |     |
| 112 | aTMS    | EQU | 62 ; ESC#8 Top margin set            | +++           |     |                       |                     |                                         |     |
| 113 | aBMS    | EQU | 63 ; ESC#2 Bottom marg set           | +++           |     |                       |                     |                                         |     |
| 114 | aSTBM   | EQU | 64 ; ESC[Pn1;Pn2r T&B margins        | DEC           |     |                       |                     |                                         |     |
| 115 | aSLRM   | EQU | 65 ; ESC[Pn1;Pn2s L&R margin         | DEC           |     |                       |                     |                                         |     |
| 116 | aCAM    | EQU | 66 ; ESC#3 Clear margins             | +++           |     |                       |                     |                                         |     |
| 117 |         |     |                                      |               |     |                       |                     |                                         |     |
| 118 | aHTS    | EQU | 67 ; ESC[H Set horiz tab             | ISO           |     |                       |                     |                                         |     |
| 119 | aVTS    | EQU | 68 ; ESC[J Set vertical tabs         | ISO           |     |                       |                     |                                         |     |

```

1 *****
2 * Commodore-Amiga, Inc. *
3 * prtbase.i *
4 *****
5 *****
6 *
7 * printer device data definition
8 *
9 *****
10
11 IFND DEVICES_PRTBASE_I
12 DEVICES_PRTBASE_I EQU 1
13
14 IFND EXEC_NODES_I
15 INCLUDE "exec/nodes.i"
16 ENDC
17 IFND EXEC_LISTS_I
18 INCLUDE "exec/lists.i"
19 ENDC
20 IFND EXEC_PORTS_I
21 INCLUDE "exec/ports.i"
22 ENDC
23 IFND EXEC_LIBRARIES_I
24 INCLUDE "exec/libraries.i"
25 ENDC
26 IFND EXEC_TASKS_I
27 INCLUDE "exec/tasks.i"
28 ENDC
29
30 IFND DEVICES_PARALLEL_I
31 INCLUDE "devices/parallel.i"
32 ENDC
33 IFND DEVICES_SERIAL_I
34 INCLUDE "devices/serial.i"
35 ENDC
36 IFND DEVICES_TIMER_I
37 INCLUDE "devices/timer.i"
38 ENDC
39 IFND LIBRARIES_DOSEXTENS_I
40 INCLUDE "libraries/dosextens.i"
41 ENDC
42 IFND INTUITION_INTUITION_I
43 INCLUDE "intuition/intuition.i"
44 ENDC
45
46
47 STRUCTURE DeviceData,LIB_SIZE
48 APTR dd_Segment ; A0 when initialized
49 APTR dd_ExecBase ; A6 for exec
50 APTR dd_CmdVectors ; command table for device commands
51 APTR dd_CmdBytes ; bytes describing which command queue
52 UWORD dd_NumCommands ; the number of commands supported
53 LABEL dd_SIZEOF
54
55
56 *-----
57 *----- device driver private variables -----
58 *-----
59 du_Flags EQU LN_PRI ; various unit flags
60
61 ;----- IO_FLAGS
62 BITDEF IO_QUEUED,4 ; command is queued to be performed
63 BITDEF IO_CURRENT,5 ; command is being performed
64 BITDEF IO_SERVICING,6 ; command is being actively performed
65 BITDEF IO_DONE,7 ; command is done
66
67 ;----- du_Flags
68 BITDEF DU_STOPPED,0 ; commands are not to be performed
69
70
71 *----- Constants -----
72 P_PRIORITY EQU 0
73 P_STKSIZE EQU $800
74
75 *----- pd_Flags -----
76 BITDEF P_IOR0,0 ; IOR0 is in use
77 BITDEF P_IOR1,1 ; IOR1 is in use
78 BITDEF P_EXPUNGED,7 ; device to be expunged when all closed
79
80 STRUCTURE PrinterData,dd_SIZEOF
81 STRUCT pd_Unit,MP_SIZE ; the one and only unit
82 BPTR pd_PrinterSegment ; the printer specific segment
83 UWORD pd_PrinterType ; the segment printer type
84 APTR pd_SegmentData ; the segment data structure
85 APTR pd_PrintBuf ; the raster print buffer
86 APTR pd_PWrite ; the parallel write function
87 APTR pd_PBothReady ; the parallel write function's done
88
89 IFGT IOEXTPar_SIZE-IOEXTSER_SIZE
90 STRUCT pd_IOR0,IOEXTPar_SIZE ; port I/O request 0
91 STRUCT pd_IOR1,IOEXTPar_SIZE ; and 1 for double buffering
92 ENDC
93
94 IFLE IOEXTPar_SIZE-IOEXTSER_SIZE
95 STRUCT pd_IOR0,IOEXTSER_SIZE ; port I/O request 0
96 STRUCT pd_IOR1,IOEXTSER_SIZE ; and 1 for double buffering
97 ENDC
98
99 STRUCT pd_TIOR,IOIV_SIZE ; timer I/O request
100 STRUCT pd_IORPort,MP_SIZE ; and message reply port
101 STRUCT pd_TC,TC_SIZE ; write task
102 STRUCT pd_Stk,P_STKSIZE ; and stack space
103 UBYTE pd_Flags ; device flags
104 UBYTE pd_pad
105 STRUCT pd_Preferences,pf_SIZEOF ; the latest preferences
106 UBYTE pd_PwaitEnabled ; wait function switch
107 LABEL pd_SIZEOF ; warning! this may be odd
108
109 BITDEF PPC GFX,0
110 BITDEF PPC_COLOR,1
111
112 PPC_BWALPHA EQU 0
113 PPC_BWGFX EQU 1
114 PPC_COLORGFX EQU 3
115
116 PCC_BW EQU 1
117 PCC_YMC EQU 2
118 PCC_YMC_BW EQU 3
119 PCC_YMCB EQU 4

```

```

120
121 STRUCTURE PrinterExtendedData,0
122 APTR ped_PrinterName ; printer name, null terminated
123 APTR ped_Init ; called after LoadSeg
124 APTR ped_Expunge ; called before UnLoadSeg
125 APTR ped_Open ; called at OpenDevice
126 APTR ped_Close ; called at CloseDevice
127 UBYTE ped_PrinterClass ; printer class
128 UBYTE ped_ColorClass ; color class
129 UBYTE ped_MaxColumns ; number of print columns available
130 UBYTE ped_NumCharSets ; number of character sets
131 UWORD ped_NumRows ; number of raster rows in a raster dump
132 ULONG ped_MaxXDots ; number of dots maximum in a raster dump
133 ULONG ped_MaxYDots ; number of dots maximum in a raster dump
134 UWORD ped_XDotsInch ; horizontal dot density
135 UWORD ped_YDotsInch ; vertical dot density
136 APTR ped_Commands ; printer text command table
137 APTR ped_DoSpecial ; special command handler
138 APTR ped_Render ; raster render function
139 LONG ped_TimeoutSecs ; good write timeout
140 LABEL ped_SIZEOF
141
142 STRUCTURE PrinterSegment,0
143 ULONG ps_NextSegment ; (actually a BPTR)
144 ULONG ps_runAlert ; MOVEQ #0,D0 : RTS
145 UWORD ps_Version ; segment version
146 UWORD ps_Revision ; segment revision
147 LABEL ps_PED ; printer extended data
148
149 ENDC

```

D - 39

```

1 *****
2 * *
3 * *
4 * *
5 * *
6 *
7 * external declarations for Serial Port Driver
8 *
9 * SOURCE CONTROL
10 * -----
11 * $Header: serial.i,v 25.0 85/03/27 19:14:15 tomp Exp $
12 *
13 * $Locker: $
14 *
15 *****
16
17 IFND DEVICES_SERIAL_I
18 DEVICES_SERIAL_I SET 1
19
20 IFND EXEC_STRINGS_I
21 include 'exec/strings.i'
22
23
24 IFND EXEC_IO_I
25 include 'exec/io.i'
26
27
28 *-----
29 *
30 * Useful constants
31 *
32 *-----
33 *
34 SER_CTL EQU $11130000 ; default char's for xON,Xoff,reserved,rsvd.
35 SER_DBAUD EQU 9600 ; default baud
36
37 *
38 *-----
39 *
40 * Driver Specific Commands
41
42 SDCMD_QUERY EQU CMD_NONSTD
43 SDCMD_BREAK EQU CMD_NONSTD+1
44 SDCMD_SETPARAMS EQU CMD_NONSTD+2
45
46 SER_DEVFINISH EQU CMD_NONSTD+2 ; number of device comands
47
48 *-----
49
50 *-- SERIALNAME is a generic macro to get the name of the driver. This
51 *-- way if the name is ever changed you will pick up the change
52 *-- automatically.
53 *--
54 *-- Normal usage would be:
55 *--
56 *-- internalName: SERIALNAME
57 *--
58
59 SERIALNAME: MACRO

```



```

60 STRING 'serial.device'
61 ENDM
62
63 BITDEF SER,XDISABLED,7 ; SERFLAGS xON-xOff feature disabled bit
64 BITDEF SER,EOFMODE,6 ; " EOF mode enabled bit
65 BITDEF SER,SHARED,5 ; " non-exclusive access
66 BITDEF SER,RAD_BOOGIE,4 ; " high-speed mode active
67 BITDEF SER,QUEUEDBRK,3 ; " queue this Break ioRqst
68 BITDEF SER,7WIRE,2 ; " RS232 7-wire protocol
69 BITDEF SER,PARTY_ODD,1 ; " parity feature enabled bit
70 BITDEF SER,PARTY_ON,0 ; " parity-enabled bit
71 BITDEF IOSER,QUEUED,6 ; IO_FLAGS rqst-queued bit
72 BITDEF IOSER,ABORT,5 ; " rqst-aborted bit
73 BITDEF IOSER,ACTIVE,4 ; " rqst-queued-or-current bit
74 BITDEF IOST,XOFFREAD,4 ; IOST_HOB receive currently xOFF'ed
75 BITDEF IOST,XOFFWRITE,3 ; " transmit currently xOFF'ed
76 BITDEF IOST,READBREAK,2 ; " break was latest input
77 BITDEF IOST,WROTEBREAK,1 ; " break was latest output
78 BITDEF IOST,OVERRUN,0 ; " status word RBF overrun
79 *
80 *****
81 STRUCTURE TERMARRAY,0
82 ULONG TERMARRAY_0
83 ULONG TERMARRAY_1
84 LABEL TERMARRAY_SIZE
85
86 *****
87 * CAUTION !! IF YOU ACCESS the serial.device, you MUST (!!!!) use an
88 * IOEXTSER-sized structure or you may overlay innocent memory, okay ?!
89 *****
90
91 STRUCTURE IOEXTSER,IOSTD_SIZE
92
93 * STRUCT MsgNode
94 * 0 APTR Succ
95 * 4 APTR Pred
96 * 8 UBYTE Type
97 * 9 UBYTE Pri
98 * A APTR Name
99 * E APTR ReplyPort
100 * 12 UWORD MNLength
101 * STRUCT IOExt
102 * 14 APTR IO_DEVICE
103 * 18 APTR IO_UNIT
104 * 1C UWORD IO_COMMAND
105 * 1E UBYTE IO_FLAGS
106 * 1F UBYTE IO_ERROR
107 * STRUCT IOSTdExt
108 * 20 ULONG IO_ACTUAL
109 * 24 ULONG IO_LENGTH
110 * 28 APTR IO_DATA
111 * 2C ULONG IO_OFFSET
112 *
113 * 30
114 ULONG IO_CTLCHAR ; control char's (order = xON,xOFF,rsvd,rsvd)
115 ULONG IO_RBUFLN ; length in bytes of serial port's read buffer
116 ULONG IO_EXTFLAGS ; (not used) flag extension area
117 ULONG IO_BAUD ; baud rate requested (true baud)
118 ULONG IO_BRKTIME ; duration of break signal in MICROseconds
119 STRUCT IO_TERMARRAY,TERMARRAY_SIZE ; termination character array

```

```

120 UBYTE IO_READLEN ; bits per read char (bit count)
121 UBYTE IO_WRITELEN ; bits per write char (bit count)
122 UBYTE IO_STOPBITS ; stopbits for read (count)
123 UBYTE IO_SERFLAGS ; see SERFLAGS bit definitions above
124 UWORD IO_STATUS ; status of serial port, as follows:
125 *
126 * BIT ACTIVE FUNCTION
127 * 0 low busy
128 * 1 low paper out
129 * 2 low select
130 * 3 low Data Set Ready
131 * 4 low Clear To Send
132 * 5 low Carrier Detect
133 * 6 low Ready To Send
134 * 7 low Data Terminal Ready
135 * 8 high read overrun
136 * 9 high break sent
137 * 10 high break received
138 * 11 high transmit x-OFFed
139 * 12 high receive x-OFFed
140 * 13-15 (not) reserved
141 *
142 LABEL IOEXTSER_SIZE
143
144 *****
145
146 *-----
147 *
148 * Driver error definitions
149 *
150 *-----
151
152 SerErr_DevBusy EQU 1
153 SerErr_BaudMismatch EQU 2
154 SerErr_InvBaud EQU 3
155 SerErr_BufErr EQU 4
156 SerErr_InvParam EQU 5
157 SerErr_LineErr EQU 6
158 SerErr_NotOpen EQU 7
159 SerErr_PortReset EQU 8
160 SerErr_ParityErr EQU 9
161 SerErr_InitErr EQU 10
162 SerErr_TimerErr EQU 11
163 SerErr_BufOverflow EQU 12
164 SerErr_NoDSR EQU 13
165 SerErr_NoCTS EQU 14
166 SerErr_DetectedBreak EQU 15
167
168

```

```

1 *****
2 * Commodore-Amiga, Inc. *
3 * timer.i *
4 *****
5 *****
6 *
7 * SOURCE CONTROL
8 * -----
9 * $Header: timer.i,v 27.1 85/06/24 13:32:40 neil Exp $
10 *
11 * $Locker: $
12 *
13 *****
14
15 IFND DEVICES_TIMER_I
16 DEVICES_TIMER_I SET 1
17
18 IFND EXEC_IO_I
19 INCLUDE "exec/io.i"
20
21
22 * unit defintions
23 UNIT_MICROHZ EQU 0
24 UNIT_VBLANK EQU 1
25
26 TIMERNAME MACRO
27 DC.B 'timer.device',0
28 DS.W 0
29 ENDM
30
31 STRUCTURE TIMEVAL,0
32 ULONG TV_SECS
33 ULONG TV_MICRO
34 LABEL TV_SIZE
35
36 STRUCTURE TIMEREQUEST,IO_SIZE
37 STRUCT IOTV_TIME,TV_SIZE
38 LABEL IOTV_SIZE
39
40 * IO_COMMAND to use for adding a timer
41 DEVINIT
42 DEVCMD TR_ADDREQUEST
43 DEVCMD TR_GETSYSTIME
44 DEVCMD TR_SETSYSTIME
45
46 ENDC

```

```

1 *****
2 * Commodore-Amiga, Inc. *
3 * trackdisk.i *
4 *****
5 *****
6 *
7 * trackdisk.i
8 *
9 * Source Control
10 * -----
11 *
12 * $Header: trackdisk.i,v 27.2 85/07/12 23:16:27 neil Exp $
13 *
14 * $Locker: $
15 *
16 *****
17
18 IFND DEVICES_TRACKDISK_I
19 DEVICES_TRACKDISK_I SET 1
20
21 IFND EXEC_IO_I
22 INCLUDE "exec/io.i"
23
24
25 -----
26 *
27 * Physical drive constants
28 *
29 -----
30
31
32 NUMCYLS EQU 80 ; normal # of cylinders
33 MAXCYLS EQU NUMCYLS+20 ; max # of cyls to look for
34 * ; during a calibrate
35 NUMSECS EQU 11
36 NUMHEADS EQU 2
37 MAXRETRY EQU 10
38 NUMTRACKS EQU NUMCYLS*NUMHEADS
39 NUMUNITS EQU 4
40
41 -----
42 *
43 * Useful constants
44 *
45 -----
46
47
48 *-- sizes before mfm encoding
49 TD_SECTOR EQU 512
50 TD_SECSHIFT EQU 9 ; log TD_SECTOR
51 * ; 2
52
53
54 -----
55 *
56 * Driver Specific Commands
57 *
58 -----
59

```

```

60 *-- TD_NAME is a generic macro to get the name of the driver. This
61 *-- way if the name is ever changed you will pick up the change
62 *-- automatically.
63 *--
64 *-- Normal usage would be:
65 *--
66 *-- internalName: TD_NAME
67 *--
68
69 TD_NAME: MACRO
70 DC.B 'trackdisk.device',0
71 DS.W 0
72 ENDM
73
74 BITDEF TD,EXTCOM,15
75
76 DEVINIT
77 DEVCMD TD_MOTOR ; control the disk's motor
78 DEVCMD TD_SEEK ; explicit seek (for testing)
79 DEVCMD TD_FORMAT ; format disk
80 DEVCMD TD_REMOVE ; notify when disk changes
81 DEVCMD TD_CHANGENUM ; number of disk changes
82 DEVCMD TD_CHANGESTATE ; is there a disk in the drive?
83 DEVCMD TD_PROTSTATUS ; is the disk write protected?
84
85 TD_LASTCOMM EQU TD_PROTSTATUS
86
87 *
88 *
89 * The disk driver has an "extended command" facility. These commands
90 * take a superset of the normal IO Request block.
91 *
92 ETD_WRITE EQU (CMD_WRITE!TDF_EXTCOM)
93 ETD_READ EQU (CMD_READ!TDF_EXTCOM)
94 ETD_MOTOR EQU (TD_MOTOR!TDF_EXTCOM)
95 ETD_SEEK EQU (TD_SEEK!TDF_EXTCOM)
96 ETD_FORMAT EQU (TD_FORMAT!TDF_EXTCOM)
97 ETD_UPDATE EQU (CMD_UPDATE!TDF_EXTCOM)
98 ETD_CLEAR EQU (CMD_CLEAR!TDF_EXTCOM)
99
100
101 *
102 * extended IO has a larger than normal io request block.
103 *
104
105 STRUCTURE IOEXTD,IOSTD_SIZE
106 ULONG IOTD_COUNT ; removal/insertion count
107 ULONG IOTD_SECLABEL ; sector label data region
108 LABEL IOTD_SIZE
109
110 * labels are TD_LABELSIZE bytes per sector
111
112 TD_LABELSIZE EQU 16
113
114 *-----
115 *
116 * Driver error defines
117 *
118 *-----
119
120 TDERR_NotSpecified EQU 20
121 TDERR_NoSecHdr EQU 21
122 TDERR_BadSecPreamble EQU 22
123 TDERR_BadSecID EQU 23
124 TDERR_BadHdrSum EQU 24
125 TDERR_BadSecSum EQU 25
126 TDERR_TooFewSecs EQU 26
127 TDERR_BadSecHdr EQU 27
128 TDERR_WriteProt EQU 28
129 TDERR_DiskChanged EQU 29
130 TDERR_SeekError EQU 30
131 TDERR_NoMem EQU 31
132 TDERR_BadUnitNum EQU 32
133 TDERR_BadDriveType EQU 33
134 TDERR_DriveInUse EQU 34
135
136

```

Contents

graphics/clip.i  
graphics/copper.i  
graphics/display.i  
graphics/gels.i  
graphics/gfx.i  
graphics/gfxbase.i  
graphics/layers.i  
graphics/rastport.i  
graphics/regions.i  
graphics/sprite.i  
graphics/text.i  
graphics/view.i

```
1 IFND GRAPHICS_CLIP_I
2 GRAPHICS_CLIP_I SET I
3 *****
4 * Commodore-Amiga, Inc.
5 * clip.i
6 *****
7
8 IFND GRAPHICS_GFX_I
9 include 'graphics/gfx.i'
10 ENDC
11 IFND EXEC_PORTS_I
12 include 'exec/ports.i'
13 ENDC
14
15 STRUCTURE Layer,0
16 LONG lr_Front
17 LONG lr_Back
18 LONG lr_ClipRect
19 LONG lr_RastPort
20 WORD lr_MinX
21 WORD lr_MinY
22 WORD lr_MaxX
23 WORD lr_MaxY
24 BYTE lr_Lock
25 BYTE lr_LockCount
26 BYTE lr_LayerLockCount
27 BYTE lr_reserved
28 WORD lr_reserved1
29 WORD lr_Flags
30 LONG lr_SuperBitMap
31 LONG lr_SuperClipRect
32 LONG lr_Window
33 WORD lr_Scroll_X
34 WORD lr_Scroll_Y
35 STRUCT lr_LockPort,MP_SIZE
36 STRUCT lr_LockMessage,MN_SIZE
37 STRUCT lr_ReplyPort,MP_SIZE
38 STRUCT lr_l_LockMessage,MN_SIZE
39 APTR lr_DamageList
40 APTR lr_cliprects
41 APTR lr_LayerInfo
42 APTR lr_LayerLocker
43 APTR lr_SuperSaverClipRects
44 APTR lr_cr
45 APTR lr_cr2
46 APTR lr_crnew
47 APTR lr_pl
48 LABEL lr_SIZEOF
49
50 STRUCTURE ClipRect,0
51 LONG cr_Next
52 LONG cr_Prev
53 LONG cr_LObs
54 LONG cr_BitMap
55 WORD cr_MinX
56 WORD cr_MinY
57 WORD cr_MaxX
58 WORD cr_MaxY
59 APTR cr_pl
```

```

60 APTR cr_p2
61 LONG cr_reserved
62 LONG cr_flags
63 LABEL cr_SIZEOF
64
65 * defines for clipping
66 ISLESSX equ 1
67 ISLESSY equ 2
68 ISGRTRX equ 4
69 ISGRTRY equ 8
70
71 ENDC

```

```

2 IFND GRAPHICS_COPPER_I
3 GRAPHICS_COPPER_I SET 1
4 *****
5 * Commodore-Amiga, Inc. *
6 * copper.i *
7 *****
8
9 COPPER_MOVE equ 0 /* pseude opcode for move #XXXX,dir */
10 COPPER_WAIT equ 1 /* pseudo opcode for wait y,x */
11 CPRNXTBUF equ 2 /* continue processing with next buffer */
12 CPR_NT_LOF equ $8000 /* copper instruction only for short frames */
13 CPR_NT_SHT equ $4000 /* copper instruction only for long frames */
14
15 STRUCTURE CopIns,0
16 WORD ci_OpCode * 0 = move, 1 = wait */
17 STRUCT ci_nxtlist,0 * UNION
18 STRUCT ci_vWaitPos,0
19 STRUCT ci_DestAddr,2
20
21 STRUCT ci_hWaitPos,0
22 STRUCT ci_DestData,2
23
24 LABEL ci_SIZEOF
25
26 * structure of cprlist that points to list that hardware actually executes */
27 STRUCTURE cprlist,0
28 APTR cpl_Next
29 APTR cpl_start
30 WORD cpl_max
31 LABEL cpl_SIZEOF
32
33 STRUCTURE CopList,0
34 APTR cl_Next /* next block for this copper list */
35 APTR cl_CopList /* system use */
36 APTR cl_ViewPort * ViewPort /* system use */
37 APTR cl_CopIns /* start of this block */
38 APTR cl_CopPtr /* intermediate ptr */
39 APTR cl_CopLStart /* mrgcop fills this in for Long Frame*/
40 APTR cl_CopSStart /* mrgcop fills this in for Short Frame*/
41 WORD cl_Count /* intermediate counter */
42 WORD cl_MaxCount /* max # of copins for this block */
43 WORD cl_DyOffset /* offset this copper list vertical waits */
44 LABEL cl_SIZEOF
45
46 STRUCTURE UCopList,0
47 APTR ucl_Next
48 APTR ucl_FirstCopList /* head node of this copper list */
49 APTR ucl_CopList /* node in use */
50 LABEL ucl_SIZEOF
51
52 * private graphics data structure
53 STRUCTURE copinit,0
54 STRUCT copinit_diagstrt,8
55 STRUCT copinit_sprstrtp,2*((2*8*2)+2+(2*2)+2)
56 STRUCT copinit_sprstop,4
57 LABEL copinit_SIZEOF
58
59 ENDC

```

```

1 IFND GRAPHICS_DISPLAY_I
2 GRAPHICS_DISPLAY_I SET 1
3 ***** display.i *****/
4 *
5 * Commodore-Amiga, Inc.
6 *
7 * Modification History
8 * date : author : Comments
9 *
10 * 8-24-84 Dale added this header file
11 *
12 *****/
13
14 * include define file for display control registers */
15 * bplcon0 defines */
16 MODE_640 equ $8000
17 PLNCNTMSK equ $7 * how many bit planes? */
18 * 0 = none, 1->6 = 1->6, 7 = reserved */
19 PLNCNTSHFT equ 12 * bits to shift for bplcon0 */
20 PF2PRI equ $40 * bplcon2 bit */
21 COLORON equ $0200 * disable color burst */
22 DBLPF equ $400
23 HOLDNMODIFY equ $800
24 INTERLACE equ 4 * interlace mode for 400 */
25
26 * bplcon1 defines */
27 PFA_FINE_SCROLL equ $F
28 PFB_FINE_SCROLL_SHIFT equ 4
29 PF_FINE_SCROLL_MASK equ $F
30
31 * display window start and stop defines */
32 DIW_HORIZ_POS equ $7F * horizontal start/stop */
33 DIW_VRTCL_POS equ $1FF * vertical start/stop */
34 DIW_VRTCL_POS_SHIFT equ 7
35
36 * Data fetch start/stop horizontal position */
37 DFTCH_MASK equ $FF
38
39 * vposr bits */
40 VPOSRL0F equ $8000
41
42 ENDC

```

```

1 IFND GRAPHICS_GELS_I
2 GRAPHICS_GELS_I SET 1
3 *****
4 * Commodore-Amiga, Inc.
5 * Graphics Library : Gels Definitions
6 *
7 *****
8
9
10 *----- VS_vSflags -----
11
12 * ;-- user-set vSprite flags --
13 SUSERFLAGS EQU $00FF ; mask of all user-settable vSprite-flags
14 BITDEF VS,VSPRITE,0 ; set if vSprite, clear if bob
15 BITDEF VS,SAVEBACK,1 ; set if background is to be saved/restored
16 BITDEF VS,OVERLAY,2 ; set to mask image of bob onto background
17 BITDEF VS,MUSTDRAW,3 ; set if vSprite absolutely must be drawn
18 * ;-- system-set vSprite flags --
19 BITDEF VS,BACKSAVED,8 ; this bob's background has been saved
20 BITDEF VS,BOBUPDATE,9 ; temporary flag, useless to outside world
21 BITDEF VS,GELGONE,10 ; set if gel is completely clipped (offscreen)
22 BITDEF VS,VSOVERFLOW,11 ; vSprite overflow (if MUSTDRAW set we draw!)
23
24
25 *----- B_flags -----
26 * ;-- these are the user flag bits --
27 BUSERFLAGS EQU $00FF ; mask of all user-settable bob-flags
28 BITDEF B,SAVEBOB,0 ; set to not erase bob
29 BITDEF B,BOBISCOMP,1 ; set to identify bob as animComp
30 * ;-- these are the system flag bits --
31 BITDEF B,BWAITING,8 ; set while bob is waiting on 'after'
32 BITDEF B,BDRAWN,9 ; set when bob is drawn this DrawG pass
33 BITDEF B,BOBSAWAY,10 ; set to initiate removal of bob
34 BITDEF B,BOBNIX,11 ; set when bob is completely removed
35 BITDEF B,SAVEPRESERVE,12 ; for back-restore during double-buffer
36 BITDEF B,OUTSTEP,13 ; for double-clearing if double-buffer
37
38
39 *----- defines for the animation procedures -----
40
41 ANFRACSIZE EQU 6
42 ANIMHALF EQU $0020
43 RINGTRIGGER EQU $0001
44
45 *----- macros -----
46 * these are GEL functions that are currently simple enough to exist as a
47 * definition. It should not be assumed that this will always be the case
48
49 InitAnimate MACRO * &animKey
50 CLR.L \1
51 ENDM
52
53
54 RemBob MACRO * &b
55 OR.W #BF_BOBSAWAY,b_BobFlags+\1
56 ENDM
57
58 *----- VS : vSprite -----
59 STRUCTURE VS,0 ; vSprite

```

```

60 * -- SYSTEM VARIABLES --
61 * GEL linked list forward/backward pointers sorted by y,x value
62 APTR vs_NextVSprite ; struct *vSprite
63 APTR vs_PrevVSprite ; struct *vSprite
64 * GEL draw list constructed in the order the bobs are actually drawn, then
65 * list is copied to clear list
66 * must be here in vSprite for system boundary detection
67 APTR vs_DrawPath ; struct *vSprite: pointer of overlay drawing
68 APTR vs_ClearPath ; struct *vSprite: pointer for overlay clearing
69 * the vSprite positions are defined in (y,x) order to make sorting
70 * sorting easier, since (y,x) as a long integer
71 WORD vs_Oldy ; previous position
72 WORD vs_Oldx ;
73 * -- COMMON VARIABLES --
74 WORD vs_VSFlags ; vSprite flags
75 * -- USER VARIABLES --
76 * the vSprite positions are defined in (y,x) order to make sorting
77 * easier, since (y,x) as a long integer
78 WORD vs_Y ; screen position
79 WORD vs_X
80 WORD vs_Height
81 WORD vs_Width ; number of words per row of image data
82 WORD vs_Depth ; number of planes of data
83 WORD vs_MeMask ; which types can collide with this vSprite
84 WORD vs_HitMask ; which types this vSprite can collide with
85 APTR vs_ImageData ; *WORD pointer to vSprite image
86 * borderLine is the one-dimensional logical OR of all
87 * the vSprite bits, used for fast collision detection of edge
88 APTR vs_BorderLine ; *WORD: logical OR of all vSprite bits
89 APTR vs_CollMask ; *WORD: similar to above except this is a
90 * matrix pointer to this vSprite's color definitions (not used by bobs)
91 APTR vs_SprColors ; *WORD
92 APTR vs_VSBob ; struct *bob: points home if this vSprite is
93 ; part of a bob
94 * planePick flag: set bit selects a plane from image, clear bit selects
95 * use of shadow mask for that plane
96 * OnOff flag: if using shadow mask to fill plane, this bit (corresponding
97 * to bit in planePick) describes whether to fill with 0's or 1's
98 * There are two uses for these flags:
99 * - if this is the vSprite of a bob, these flags describe how
100 * the bob is to be drawn into memory
101 * - if this is a simple vSprite and the user intends on setting
102 * the MUSTDRAW flag of the vSprite, these flags must be set
103 * too to describe which color registers the user wants for
104 * the image
105 BYTE vs_PlanePick
106 BYTE vs_PlaneOnOff
107 LABEL vs_UserExt ; user definable
108 LABEL vs_SIZEOF
109
110
111 *----- BOB : bob -----
112
113 STRUCTURE BOB,0 ; bob: blitter object
114 * -- COMMON VARIABLES --
115 APTR bob_SavePlanes ; * *WORD for each plane in RastPort
116 WORD bob_BobFlags ; general purpose flags (see definitions below)
117 * -- USER VARIABLES --
118 APTR bob_SaveBuffer ; *WORD pointer to the buffer for background
119 * save used by bobs for "cookie-cutting" and multi-plane masking
120 APTR bob_ImageShadow ; *WORD
121 * pointer to BOBs for sequenced drawing of bobs
122 * for correct overlaying of multiple component animations
123 APTR bob_Before ; struct *bob: draw this bob before bob pointed
124 ; to by before
125 APTR bob_After ; struct *bob: draw this bob after bob pointed
126 ; to by after
127 APTR bob_BobVSprite ; struct *vSprite: this bob's vSprite definitio
128 APTR bob_BobComp ; struct *animComp: pointer to this bob's
129 ; animComp def
130 APTR bob_DBuffer ; struct dBufPacket: pointer to this bob's
131 ; dBuf packet
132 LABEL bob_BUserExt ; bob user extension
133 LABEL bob_SIZEOF
134
135 *----- AC : animComp -----
136
137 STRUCTURE AC,0 ; animComp
138 * -- COMMON VARIABLES --
139 WORD ac_CompFlags ; animComp flags for system & user
140 * timer defines how long to keep this component active:
141 * if set non-zero, timer decrements to zero then switches to nextSeq
142 * if set to zero, animComp never switches
143 WORD ac_Timer
144 * -- USER VARIABLES --
145 * initial value for timer when the animComp is activated by the system
146 WORD ac_TimeSet
147 * pointer to next and previous components of animation object
148 APTR ac_NextComp ; struct *animComp
149 APTR ac_PrevComp ; struct *animComp
150 * pointer to component component definition of next image in sequence
151 APTR ac_NextSeq ; struct *animComp
152 APTR ac_PrevSeq ; struct *animComp
153 APTR ac_AnimCRoutine ; address of special animation procedure
154 WORD ac_YTrans ; initial y translation (if this is a component
155 WORD ac_XTrans ; initial x translation (if this is a component
156 APTR ac_HeadOb ; struct *animOb
157 APTR ac_AnimBob ; struct *bob
158 LABEL ac_SIZE
159
160 *----- AO : animOb -----
161
162 STRUCTURE AO,0 ; animOb
163 * -- SYSTEM VARIABLES --
164 APTR ao_NextOb ; struct *animOb
165 APTR ao_PrevOb ; struct *animOb
166 * number of calls to Animate this animOb has endured
167 LONG ao_Clock
168 WORD ao_AnOldY ; old y,x coordinates
169 WORD ao_AnOldX ;
170 * -- COMMON VARIABLES --
171 WORD ao_AnY ; y,x coordinates of the animOb
172 WORD ao_AnX ;
173 * -- USER VARIABLES --
174 WORD ao_YVel ; velocities of this object
175 WORD ao_XVel ;
176 WORD ao_XAccel ; accelerations of this object
177 WORD ao_YAccel ; !!! backwards !!!
178 WORD ao_RingYTrans ; ring translation values
179 WORD ao_RingXTrans ;

```

```

180 APTR ao_AnimORoutine ; address of special animation procedure
181 APTR ao_HeadComp ; struct *animComp: pointer to first component
182 LABEL ao_AUserExt ; animOb user extension
183 LABEL ao_SIZEOF
184
185
186 *----- DBP : dBufPacket -----
187 * dBufPacket defines the values needed to be saved across buffer to buffer
188 * when in double-buffer mode
189
190 STRUCTURE DBP,0 ; dBufPacket
191 WORD dbp_BufY ; save the other buffers screen coordinates
192 WORD dbp_BufX ;
193 APTR dbp_BufPath ; struct *vSprite: carry the draw path over
194 ; the gap
195 * these pointers must be filled in by the user
196 * pointer to other buffer's background save buffer
197 APTR dbp_BufBuffer ; *WORD
198 * pointer to other buffer's background plane pointers
199 APTR dbp_BufPlanes ; **WORD
200 LABEL dbp_SIZEOF
201
202 ENDC

```

```

1 *****
2 *
3 * Commodore-Amiga, Inc.
4 * gfx.i
5 *
6 *****
7 IFND GRAPHICS_GFX_I
8 GRAPHICS_GFX_I SET 1
9
10 BITSET equ $8000
11 BITCLR equ 0
12 AGNUS equ 1
13 DENISE equ 1
14
15 STRUCTURE BitMap,0
16 WORD bm_BytesPerRow
17 WORD bm_Rows
18 BYTE bm_Flags
19 BYTE bm_Depth
20 WORD bm_Pad
21 STRUCT bm_Planes,8*4
22 LABEL bm_SIZEOF
23
24 STRUCTURE Rectangle,0
25 WORD ra_MinX
26 WORD ra_MinY
27 WORD ra_MaxX
28 WORD ra_MaxY
29 LABEL ra_SIZEOF
30
31 ENDC

```



```

1 ***** gfxbase.i *****
2 *
3 * Commodore-Amiga, Inc.
4 *
5 *****
6 IFND GRAPHICS_GFXBASE_I
7 GRAPHICS_GFXBASE_I SET 1
8
9 IFND EXEC_LISTS_I
10 include 'exec/lists.i'
11 ENDC
12 IFND EXEC_LIBRARIES_I
13 include 'exec/libraries.i'
14 ENDC
15 IFND EXEC_INTERRUPTS_I
16 include 'exec/interrupts.i'
17 ENDC
18
19 STRUCTURE GfxBase,LIB_SIZE
20 APTR gb_ActiView ; struct *View
21 APTR gb_copinit ; struct *copinit ; ptr to copper start up list
22 APTR gb_cia ; for 6526 resource use
23 APTR gb_blitter ; for blitter resource use
24 APTR gb_LOFlist ; current copper list being run
25 APTR gb_SHFlist ; current copper list being run
26 APTR gb_blthd ; struct *bltnode
27 APTR gb_blttl ;
28 APTR gb_bsblthd ;
29 APTR gb_bsblttl ;
30 STRUCT gb_vbsrv,IS_SIZE
31 STRUCT gb_timsrv,IS_SIZE
32 STRUCT gb_bltsrv,IS_SIZE
33 STRUCT gb_TextFonts,LH_SIZE
34 APTR gb_DefaultFont
35 UWORD gb_Modes ; copy of bltcon0
36 BYTE gb_VBlank
37 BYTE gb_Debug
38 UWORD gb_BeamSync
39 WORD gb_system_bplcon0
40 BYTE gb_SpriteReserved
41 BYTE gb_bytereserved
42
43 WORD gb_Flags
44 WORD gb_BlitLock
45 WORD gb_BlitNest
46 STRUCT gb_BlitWaitQ,LH_SIZE
47 APTR gb_BlitOwner
48 STRUCT gb_TOF_WaitQ,LH_SIZE
49 WORD gb_DisplayFlags
51 STRUCT gb_reserved,8 ; 8 bytes reserved for future use
52 LABEL gb_SIZE
53
54 * bits for dalestuff, which may go away when blitter becomes a resource
55 OWNBLITTERN equ 0 * blitter owned bit
56 QBOWNERN equ 1 * blitter owned by blit queuer
58 QBOWNER equ 1<<QBOWNERN
59
60 ENDC

```

```

2 * Commodore-Amiga, Inc.
3 * layers.i
4 *
5 IFND GRAPHICS_LAYERS_I
6 GRAPHICS_LAYERS_I SET 1
7
8 IFND EXEC_PORTS_I
9 include 'exec/ports.i'
10 ENDC
11
12 IFND EXEC_LISTS_I
13 include 'exec/lists.i'
14 ENDC
15
16 STRUCTURE LayerInfo_extra,0
17 STRUCT lie_env,l3*4
18 STRUCT lie_mem,LH_SIZE
19 APTR lie_FreeClipRects
20 APTR lie_blitbuff
21 LABEL lie_SIZEOF
22
23 LMN_REGION equ -1
24
25 STRUCTURE mem_node,0
26 APTR memnode_succ
27 APTR memnode_pred
28 APTR memnode_where
29 LONG memnode_how_big
30 LABEL memnode_SIZEOF
31
32 STRUCTURE Layer_Info,0
33 APTR li_top_layer
34 APTR li_check_lp
35 APTR li_obs
36 STRUCT li_RP_ReplyPort,MP_SIZE
37 STRUCT li_LockPort,MP_SIZE
38 BYTE li_Lock
39 BYTE li_broadcast
40 BYTE li_locknest
41 BYTE li_pad
42 APTR li_Locker
43 STRUCT li_bytereserved,2
44 STRUCT li_wordreserved,4
45 STRUCT li_longreserved,4
46 APTR li_LayerInfo_extra
47 LABEL li_SIZEOF
48
49 NEWLAYERINFO_CALLED equ 1
50
51 ENDC

```

```

1 ***** rastport.i *****
2 *
3 * Commodore-Amiga, Inc.
4 *
5 *****
6 IFND GRAPHICS_RASTPORT_I
7 GRAPHICS_RASTPORT_I SET 1
8
9 IFND GRAPHICS_GFX_I
10 include 'graphics/gfx.i'
11 ENDC
12
13 *----- TR : TmpRas -----
14
15 STRUCTURE TmpRas,0
16 APTR tr_RasPtr ; *WORD
17 LONG tr_Size
18 LABEL tr_SIZEOF
19
20 *----- GelsInfo
21
22 STRUCTURE GelsInfo,0
23 BYTE gi_sprRsrvd * flag of which sprites to reserve from
24 * vsprite system
25 BYTE gi_Flags * reserved for system use
26 APTR gi_gelHead
27 APTR gi_gelTail * dummy vSprites for list management
28 * pointer to array of 8 WORDS for sprite available lines
29 APTR gi_nextLine
30 * pointer to array of 8 pointers for color-last-assigned to vSprites
31 APTR gi_lastColor
32 APTR gi_collHandler * addresses of collision routines
33 SHORT gi_leftmost
34 SHORT gi_rightmost
35 SHORT gi_topmost
36 SHORT gi_bottommost
37 APTR gi_firstBlissObj
38 APTR gi_lastBlissObj * system use only
39 LABEL gi_SIZEOF
40
41 *----- RP_Flags -----
42 BITDEF RP,FRST_DOT,0 ; draw the first dot of this line ?
43 BITDEF RP,ONE_DOT,1 ; use one dot mode for drawing lines
44 BITDEF RP,DBUFFER,2 ; flag set when RastPorts are double-buffered
45 * ; (only used for bobs)
46 BITDEF RP,AREAOUTLINE,3 ; used by areafiller
47 BITDEF RP,NOCROSSFILL,5 ; used by areafiller
48
49 *----- RP_DrawMode -----
50 RP_JAM1 EQU 0
51 RP_JAM2 EQU 1
52 RP_COMPLEMENT EQU 2
53 RP_INVERSVID EQU 4 ; inverse video for drawing modes
54
55 *----- RP_TxFlags -----
56 BITDEF RP,TXSCALE,0
57
58 STRUCTURE RastPort,0
59 LONG rp_Layer

```

```

60 LONG rp_BitMap
61 LONG rp_AreaPtrn
62 LONG rp_TmpRas
63 LONG rp_AreaInfo
64 LONG rp_GelsInfo
65 BYTE rp_Mask
66 BYTE rp_FgPen
67 BYTE rp_BgPen
68 BYTE rp_AOLPen
69 BYTE rp_DrawMode
70 BYTE rp_AreaPtSz
71 BYTE rp_Dummy
72 BYTE rp_linpatcnt
73 WORD rp_Flags
74 WORD rp_LinePtrn
75 WORD rp_cp_x
76 WORD rp_cp_y
77 STRUCT rp_minterms,8
78 WORD rp_PenWidth
79 WORD rp_PenHeight
80 LONG rp_Font
81 BYTE rp_AlgoStyle
82 BYTE rp_TxFlags
83 WORD rp_TxHeight
84 WORD rp_TxWidth
85 WORD rp_TxBaseline
86 WORD rp_TxSpacing
87 APTR rp_RP_User
88 STRUCT rp_wordreserved,14
89 STRUCT rp_longreserved,8
90 STRUCT rp_reserved,8
91 LABEL rp_SIZEOF
92
93 STRUCTURE AreaInfo,0
94 LONG ai_VctrTbl
95 LONG ai_VctrPtr
96 LONG ai_FlagTbl
97 LONG ai_FlagPtr
98 WORD ai_Count
99 WORD ai_MaxCount
100 WORD ai_FirstX
101 WORD ai_FirstY
102 LABEL ai_SIZEOF
103
104 ONE_DOTn equ 1
105 ONE_DOT equ $2 * 1<<ONE_DOTn
106 FRST_DOTn equ 0
107 FRST_DOT equ 1 * 1<<FRST_DOTn
108
109 ENDC

```

```

1
2 IFND GRAPHICS_REGIONS_I
3 GRAPHICS_REGIONS_I SET 1
4 *****
5 * Commodore-Amiga, Inc. *
6 * regions.i *
7 *****
8
9 IFND GRAPHICS_GFX_I
10 include 'graphics/gfx.i'
11 ENDC
12
13 STRUCTURE Region,0
14 STRUCT rg_bounds,ra_SIZEOF
15 APTR rg_RegionRectangle
16 LABEL rg_SIZEOF
17
18 STRUCTURE RegionRectangle,0
19 APTR rr_Next
20 APTR rr_Prev
21 STRUCT rr_bounds,ra_SIZEOF
22 LABEL rr_SIZEOF
23
24 ENDC

```

```

1 IFND GRAPHICS_SPRITE_I
2 GRAPHICS_SPRITE_I SET 1
3 *****
4 * Commodore-Amiga, Inc. *
5 * sprite.h *
6 *****
7
8 STRUCTURE SimpleSprite,0
9 APTR ss_posctldata
10 WORD ss_height
11 WORD ss_x
12 WORD ss_y
13 WORD ss_num
14 LABEL ss_SIZEOF
15
16 ENDC

```

```

1 IFND GRAPHICS_TEXT_I
2 GRAPHICS_TEXT_I SET 1
3 *****
4 * Commodore-Amiga, Inc. *
5 * text.i *
6 *****
7 *****
8 *
9 * graphics library text structures
10 *
11 *****
12
13 IFND EXEC_PORTS_I
14 INCLUDE "exec/ports.i"
15 ENDC
16
17 *----- Font Styles -----
18 FS_NORMAL EQU 0 ;normal text (no style attributes set)
19 BITDEF FS,EXTENDED,3 ;extended face (must be designed)
20 BITDEF FS,ITALIC,2 ;italic (slanted 1:2 right)
21 BITDEF FS,BOLD,1 ;bold face text (ORED w/ shifted right 1)
22 BITDEF FS,UNDERLINED,0 ;underlined (under baseline)
23
24 *----- Font Flags -----
25 BITDEF FP,ROMFONT,0 ;font is in rom
26 BITDEF FP,DISKFONT,1 ;font is from diskfont.library
27 BITDEF FP,REVPATH,2 ;designed path is reversed (e.g. left)
28 BITDEF FP,TALLDOT,3 ;designed for hires non-interlaced
29 BITDEF FP,WIDEDOT,4 ;designed for lores interlaced
30 BITDEF FP,PROPORTIONAL,5 ;character sizes can vary from nominal
31 BITDEF FP,DESIGNED,6 ;size is "designed", not constructed
32 BITDEF FP,REMOVED,7 ; the font has been removed
33
34
35 ***** TextAttr node *****
36 STRUCTURE TextAttr,0
37 APTR ta_Name ;name of the desired font
38 UWORD ta_YSize ;size of the desired font
39 UBYTE ta_Style ;desired font style
40 UBYTE ta_Flags ;font preferences
41 LABEL ta_SIZEOF
42
43
44 ***** TextFont node *****
45 STRUCTURE TextFont,MN_SIZE
46 *
47 UWORD tf_YSize ;font name in LN \ used in this
48 UBYTE tf_Style ;font height } order to best
49 UBYTE tf_Flags ;font style } match a font
50 UWORD tf_XSize ;preference attributes / request.
51 UWORD tf_Baseline ;nominal font width
52 UWORD tf_BoldSmear ;distance from the top of char to baseline
53 ;smear to affect a bold enhancement
54
55 UWORD tf_Accessors ;access count
56
57 UBYTE tf_LoChar ;the first character described here
58 UBYTE tf_HiChar ;the last character described here
59 APTR tf_CharData ;the bit character data

```

```

60 UWORD tf_Modulo ;the row modulo for the strike font data
61 APTR tf_CharLoc ;ptr to location data for the strike font
62 * ; 2 words: bit offset then size
63 APTR tf_CharSpace ;ptr to words of proportional spacing data
64 APTR tf_CharKern ;ptr to words of kerning data
65 LABEL tf_SIZEOF
66
67 ENDC

```

```

1 IFND GRAPHICS_VIEW_I
2 GRAPHICS_VIEW_I SET 1
3 *****
4 * Commodore-Amiga, Inc. *
5 * view.i *
6 *****
7
8 IFND GRAPHICS_GFX_I
9 include 'graphics/gfx.i'
10 ENDC
11
12 V_PFBA EQU $40
13 V_DUALPF EQU $400
14 V_HIRES EQU $8000
15 V_LACE EQU 4
16 V_HAM EQU $800
17 V_SPRITES EQU $4000
18 GENLOCK_VIDEO EQU 2
19
20 STRUCTURE ColorMap,0
21 BYTE cm_Flags
22 BYTE cm_Type
23 WORD cm_Count
24 APTR cm_ColorTable
25 LABEL cm_SIZEOF
26
27
28 STRUCTURE ViewPort,0
29 LONG vp_Next
30 LONG vp_ColorMap
31 LONG vp_DspIns
32 LONG vp_SprIns
33 LONG vp_ClrIns
34 LONG vp_UCopIns
35 WORD vp_DWidth
36 WORD vp_DHeight
37 WORD vp_DxOffset
38 WORD vp_DyOffset
39 WORD vp_Modes
40 WORD vp_reserved
41 APTR vp_RasInfo
42 LABEL vp_SIZEOF
43
44
45 STRUCTURE View,0
46 LONG v_ViewPort
47 LONG v_LOFCprList
48 LONG v_SHFCprList
49 WORD v_DyOffset
50 WORD v_DxOffset
51 WORD v_Modes
52 LABEL v_SIZEOF
53
54
55 STRUCTURE collTable,0
56 LONG cp_collPtrs,16
57 LABEL cp_SIZEOF
58
59
60 STRUCTURE RasInfo,0
61 APTR ri_Next
62 LONG ri_BitMap
63 WORD ri_RxOffset
64 WORD ri_RyOffset
65 LABEL ri_SIZEOF
66
67 ENDC

```

Contents

hardware/adkbits.i  
hardware/blit.i  
hardware/cia.i  
hardware/custom.i  
hardware/dmabits.i  
hardware/intbits.i

```
1 *****
2 * adkbits.i -- bit definitions for adkcon register
3 *
4 * Commodore-Amiga, Inc.
5 *
6 * $Header: adkbits.i,v 27.1 85/06/24 14:42:37 neil Exp $
7 *
8 * $Locker: $
9 *
10 *****
11
12 IFND HARDWARE_ADKBITS_I
13 HARDWARE_ADKBITS_I SET 1
14
15 ADKB_SETCLR EQU 15 ; standard set/clear bit
16 ADKB_PRECOMP1 EQU 14 ; two bits of precompensation
17 ADKB_PRECOMP0 EQU 13
18 ADKB_MFMPREC EQU 12 ; use mfm style precompensation
19 ADKB_UARTBRK EQU 11 ; force uart output to zero
20 ADKB_WORDSYNC EQU 10 ; enable DSKSYNC register matching
21 ADKB_MSBSYNC EQU 9 ; (Apple GCR Only) sync on MSB for reading
22 ADKB_FAST EQU 8 ; 1 -> 2 us/bit (mfm), 2 -> 4 us/bit (gcr)
23 ADKB_USE3PN EQU 7 ; use aud chan 3 to modulate period of ??
24 ADKB_USE2P3 EQU 6 ; use aud chan 2 to modulate period of 3
25 ADKB_USE1P2 EQU 5 ; use aud chan 1 to modulate period of 2
26 ADKB_USE0P1 EQU 4 ; use aud chan 0 to modulate period of 1
27 ADKB_USE3VN EQU 3 ; use aud chan 3 to modulate volume of ??
28 ADKB_USE2V3 EQU 2 ; use aud chan 2 to modulate volume of 3
29 ADKB_USE1V2 EQU 1 ; use aud chan 1 to modulate volume of 2
30 ADKB_USE0V1 EQU 0 ; use aud chan 0 to modulate volume of 1
31
32 ADKF_SETCLR EQU (1<<15)
33 ADKF_PRECOMP1 EQU (1<<14)
34 ADKF_PRECOMP0 EQU (1<<13)
35 ADKF_MFMPREC EQU (1<<12)
36 ADKF_UARTBRK EQU (1<<11)
37 ADKF_WORDSYNC EQU (1<<10)
38 ADKF_MSBSYNC EQU (1<<9)
39 ADKF_FAST EQU (1<<8)
40 ADKF_USE3PN EQU (1<<7)
41 ADKF_USE2P3 EQU (1<<6)
42 ADKF_USE1P2 EQU (1<<5)
43 ADKF_USE0P1 EQU (1<<4)
44 ADKF_USE3VN EQU (1<<3)
45 ADKF_USE2V3 EQU (1<<2)
46 ADKF_USE1V2 EQU (1<<1)
47 ADKF_USE0V1 EQU (1<<0)
48
49 ADKF_PRE000NS EQU 0 ; 000 ns of precomp
50 ADKF_PRE140NS EQU (ADKF_PRECOMP0) ; 140 ns of precomp
51 ADKF_PRE280NS EQU (ADKF_PRECOMP1) ; 280 ns of precomp
52 ADKF_PRE560NS EQU (ADKF_PRECOMP0!ADKF_PRECOMP1) ; 560 ns of precomp
53
54
```

```

1 *****
2 * Commodore-Amiga, Inc.
3 * blit.i
4 *
5 * $Header: blit.i,v 27.1 85/06/24 14:42:42 neil Exp $
6 *
7 * $Locker: $
8 *
9 *****
10
11 IFND HARDWARE_BLIT_I
12 HARDWARE_BLIT_I SET 1
13
14 STRUCTURE bltnode,0
15 LONG bn_n
16 LONG bn_function
17 BYTE bn_stat
18 BYTE bn_dummy
19 WORD bn_blitsize
20 WORD bn_beamsync
21 LONG bn_cleanup
22 LABEL bn_SIZEOF
23
24 * bit defines used by blit queuer
25 CLEANMEN equ 6
26 CLEANME equ 1<<CLEANMEN
27
28 * include file for blitter */
29 HSIZEBITS equ 6
30 VSIZEBITS equ 16-HSIZEBITS
31 HSIZEMASK equ $3f /* 2^6 - 1 */
32 VSIZEMASK equ $3FF /* 2^10 - 1 */
33
34 MAXBYTESPERROW EQU 128
35
36 * definitions for blitter control register 0 */
37
38 ABC equ $80
39 ABNC equ $40
40 ANBC equ $20
41 ANBNC equ $10
42 NABC equ $8
43 NABNC equ $4
44 NANBC equ $2
45 NANBNC equ $1
46
47 BCOB_DEST equ 8
48 BCOB_SRCC equ 9
49 BCOB_SRCB equ 10
50 BCOB_SRCA equ 11
51 BCOF_DEST equ $100
52 BCOF_SRCC equ $200
53 BCOF_SRCB equ $400
54 BCOF_SRCA equ $800
55
56 BCLF_DESC equ 2
57
58 DEST equ $100
59 SRCC equ $200
60 SRCB equ $400
61 SRCA equ $800
62
63 ASHIFTSHIFT equ 12 /* bits to right align ashift value */
64 BSHIFTSHIFT equ 12 /* bits to right align bshift value */
65
66 * definations for blitter control register 1 */
67 LINEMODE equ $1
68 FILL_OR equ $8
69 FILL_XOR equ $10
70 FILL_CARRYIN equ $4
71 ONEDOT equ $2
72 OVFLAG equ $20
73 SIGNFLAG equ $40
74 BLITREVERSE equ $2
75
76 SUD equ $10
77 SUL equ $8
78 AUL equ $4
79
80 OCTANT8 equ 24
81 OCTANT7 equ 4
82 OCTANT6 equ 12
83 OCTANT5 equ 28
84 OCTANT4 equ 20
85 OCTANT3 equ 8
86 OCTANT2 equ 0
87 OCTANT1 equ 16
88
89

```

```

1: *****
2: * Commodore-Amiga, Inc.
3: * cia.i -- definitions for the registers and bits in the
 Complex Interface
4: * Adapter (CIA) chip
5: * $Header: cia.i,v 27.1 85/06/24 14:42:49 neil Exp $
6: * $Locker: $
7: *
8: *****
9: IFND HARDWARE_CIA_I
10: HARDWARE_CIA_I SET 1
11: *
12: * _ciaa is on an ODD address (e.g. the low byte) -- $bfe001
13: * _ciab is on an EVEN address (e.g. the high byte) -- $bfd000
14: *
15: * do this to get the definitions:
16: * XREF _ciaa
17: * XREF _ciab
18: *
19: * cia register offsets
20: ciapra EQU $0000
21: ciaprb EQU $0100
22: ciaddr EQU $0200
23: ciaddrb EQU $0300
24: ciatalo EQU $0400
25: ciatahi EQU $0500
26: ciatblo EQU $0600
27: ciatbhi EQU $0700
28: ciatodlow EQU $0800
29: ciatodmid EQU $0900
30: ciatodhi EQU $0A00
31: ciasdr EQU $0C00
32: ciaicr EQU $0D00
33: ciacra EQU $0E00
34: ciacrb EQU $0F00
35:
36: * interrupt control register bit numbers
37: CIAICRB_TA EQU 0
38: CIAICRB_TB EQU 1
39: CIAICRB_ALRM EQU 2
40: CIAICRB_SP EQU 3
41: CIAICRB_FLG EQU 4
42: CIAICRB_IR EQU 7
43: CIAICRB_SETCLR EQU 7
44:
45: * control register A bit numbers
46: CIACRAB_START EQU 0
47: CIACRAB_PBon EQU 1
48: CIACRAB_OUTMODE EQU 2
49: CIACRAB_RUNMODE EQU 3
50: CIACRAB_LOAD EQU 4
51: CIACRAB_INMODE EQU 5
52: CIACRAB_SPMODE EQU 6
53: CIACRAB_TODIN EQU 7
54:
55: * control register B bit numbers
56: CIACRBB_START EQU 0
57: CIACRBB_PBon EQU 1
58: CIACRBB_OUTMODE EQU 2
59:
60:
61:
62:
63:
64:
65: CIACRBB_RUNMODE EQU 3
66: CIACRBB_LOAD EQU 4
67: CIACRBB_INMODE0 EQU 5
68: CIACRBB_INMODE1 EQU 6
69: CIACRBB_ALARM EQU 7
70:
71: * interrupt control register bit masks
72: CIAICRF_TA EQU (1<<0)
73: CIAICRF_TB EQU (1<<1)
74: CIAICRF_ALRM EQU (1<<2)
75: CIAICRF_SP EQU (1<<3)
76: CIAICRF_FLG EQU (1<<4)
77: CIAICRF_IR EQU (1<<7)
78: CIAICRF_SETCLR EQU (1<<7)
79:
80: * control register A bit masks
81: CIACRAF_START EQU (1<<0)
82: CIACRAF_PBon EQU (1<<1)
83: CIACRAF_OUTMODE EQU (1<<2)
84: CIACRAF_RUNMODE EQU (1<<3)
85: CIACRAF_LOAD EQU (1<<4)
86: CIACRAF_INMODE EQU (1<<5)
87: CIACRAF_SPMODE EQU (1<<6)
88: CIACRAF_TODIN EQU (1<<7)
89:
90: * control register B bit masks
91: CIACRBF_START EQU (1<<0)
92: CIACRBF_PBon EQU (1<<1)
93: CIACRBF_OUTMODE EQU (1<<2)
94: CIACRBF_RUNMODE EQU (1<<3)
95: CIACRBF_LOAD EQU (1<<4)
96: CIACRBF_INMODE0 EQU (1<<5)
97: CIACRBF_INMODE1 EQU (1<<6)
98: CIACRBF_ALARM EQU (1<<7)
99:
100: * control register B INMODE masks
101: CIACRBF_IN_PHI2 EQU 0
102: CIACRBF_IN_CNT EQU (CIACRBF_INMODE0)
103: CIACRBF_IN_TA EQU (CIACRBF_INMODE1)
104: CIACRBF_IN_CNT_TA EQU (CIACRBF_INMODE0!CIACRBF_INMODE1)
105:
106:
107: *
108: * Port definitions -- what each bit in a cia peripheral
 register is tied to
109: *
110:
111: * ciaa port A (0xbfe001)
112: CIAB_GAMEPORT1 EQU (7) * gameport 1, pin 6 (fire
 button*)
113: CIAB_GAMEPORT0 EQU (6) * gameport 0, pin 6 (fire
 button*)
114: CIAB_DSKRDY EQU (5) * disk ready*
115: CIAB_DSKTRACK0 EQU (4) * disk on track 00*
116: CIAB_DSKPROT EQU (3) * disk write protect*
117: CIAB_DSKCHANGE EQU (2) * disk change*
118: CIAB_LED EQU (1) * led light control (0==>bright)
119: CIAB_OVERLAY EQU (0) * memory overlay bit
120:
121: * ciaa port B (0xbfef01) -- parallel port
122:

```



```

123: * ciab port A (0xbfd000) -- serial and printer control
124: CIAB_COMDTR EQU (7) * serial Data Terminal Ready*
125: CIAB_COMRTS EQU (6) * serial Request to Send*
126: CIAB_COMCD EQU (5) * serial Carrier Detect*
127: CIAB_COMCTS EQU (4) * serial Clear to Send*
128: CIAB_COMDSR EQU (3) * serial Data Set Ready*
129: CIAB_PRTRSEL EQU (2) * printer SELECT
130: CIAB_PRTRPOUT EQU (1) * printer paper out
131: CIAB_PRTRBUSY EQU (0) * printer busy
132:
133: * ciab port B (0xbfd100) -- disk control
134: CIAB_DSKMOTOR EQU (7) * disk motorr*
135: CIAB_DSKSEL3 EQU (6) * disk select unit 3*
136: CIAB_DSKSEL2 EQU (5) * disk select unit 2*
137: CIAB_DSKSEL1 EQU (4) * disk select unit 1*
138: CIAB_DSKSEL0 EQU (3) * disk select unit 0*
139: CIAB_DSKSIDE EQU (2) * disk side select*
140: CIAB_DSKDIREC EQU (1) * disk direction of seek*
141: CIAB_DSKSTEP EQU (0) * disk step heads*
142:
143: * ciao port A (0xbfe001)
144: CIAF_GAMEPORT1 EQU (1<<7)
145: CIAF_GAMEPORT0 EQU (1<<6)
146: CIAF_DSKRDY EQU (1<<5)
147: CIAF_DSKTRACK0 EQU (1<<4)
148: CIAF_DSKPROT EQU (1<<3)
149: CIAF_DSKCHANGE EQU (1<<2)
150: CIAF_LED EQU (1<<1)
151: CIAF_OVERLAY EQU (1<<0)
152:
153: * ciao port B (0xbfe101) -- parallel port
154:
155: * ciab port A (0xbfd000) -- serial and printer control
156: CIAF_COMDTR EQU (1<<7)
157: CIAF_COMRTS EQU (1<<6)
158: CIAF_COMCD EQU (1<<5)
159: CIAF_COMCTS EQU (1<<4)
160: CIAF_COMDSR EQU (1<<3)
161: CIAF_PRTRSEL EQU (1<<2)
162: CIAF_PRTRPOUT EQU (1<<1)
163: CIAF_PRTRBUSY EQU (1<<0)
164:
165: * ciab port B (0xbfd100) -- disk control
166: CIAF_DSKMOTOR EQU (1<<7)
167: CIAF_DSKSEL3 EQU (1<<6)
168: CIAF_DSKSEL2 EQU (1<<5)
169: CIAF_DSKSEL1 EQU (1<<4)
170: CIAF_DSKSEL0 EQU (1<<3)
171: CIAF_DSKSIDE EQU (1<<2)
172: CIAF_DSKDIREC EQU (1<<1)
173: CIAF_DSKSTEP EQU (1<<0)
174:
175: ENDC !HARDWARE_CIA_I

```

```

1 *****
2 * Commodore-Amiga, Inc.
3 * custom.i
4
5 * $Header: custom.i,v 27.1 85/06/24 14:42:56 neil Exp $
6 * $Locker: $
7 *
8 *
9 *****
11 IFND HARDWARE_CUSTOM_I
12 HARDWARE_CUSTOM_I SET 1
13
14 *
15 * do this to get base of custom registers:
16 * XREF _custom;
17 *
18
19 bltddat EQU $000
20 dmaconr EQU $002
21 vposr EQU $004
22 vhposr EQU $006
23 dskdatr EQU $008
24 joy0dat EQU $00A
25 joyldat EQU $00C
26 clxdat EQU $00E
27
28 adkconr EQU $010
29 pot0dat EQU $012
30 potldat EQU $014
31 potinp EQU $016
32 serdatr EQU $018
33 dskbytr EQU $01A
34 intenar EQU $01C
35 intreqr EQU $01E
36
37 dskpt EQU $020
38 dsklen EQU $024
39 dskdat EQU $026
40 refptr EQU $028
41 vposw EQU $02A
42 vhposw EQU $02C
43 copcon EQU $02E
44 serdat EQU $030
45 serper EQU $032
46 potgo EQU $034
47 joytest EQU $036
48 strequ EQU $038
49 strvbl EQU $03A
50 strhor EQU $03C
51 strlong EQU $03E
52
53 bltcon0 EQU $040
54 bltcon1 EQU $042
55 bltafwm EQU $044
56 bltalwm EQU $046
57 bltcept EQU $048
58 bltbpt EQU $04C
59 bltapt EQU $050
60 bltdpt EQU $054
61 bltsize EQU $058
62
63 bltcm0d EQU $060

```

```

64 bltbmod EQU $062
65 bltamod EQU $064
66 bltdmod EQU $066
67
68 bltcdat EQU $070
69 bltbdat EQU $072
70 bltadat EQU $074
71
72 dsksync EQU $07E
73
74 copllc EQU $080
75 cop2lc EQU $084
76 copjmpl EQU $088
77 copjmp2 EQU $08A
78 copins EQU $08C
79 diwstrt EQU $08E
80 diwstop EQU $090
81 ddfstrt EQU $092
82 ddfstop EQU $094
83 dmacon EQU $096
84 clxcon EQU $098
85 intena EQU $09A
86 intreq EQU $09C
87 adkcon EQU $09E
88
89 aud EQU $0A0
90 aud0 EQU $0A0
91 aud1 EQU $0B0
92 aud2 EQU $0C0
93 aud3 EQU $0D0
94
95 * STRUCTURE AudChannel,0
96 ac_ptr EQU $00 ; ptr to start of waveform data
97 ac_len EQU $04 ; length of waveform in words
98 ac_per EQU $06 ; sample period
99 ac_vol EQU $08 ; volume
100 ac_dat EQU $0A ; sample pair
101 ac_SIZEOF EQU $10
102
103 bplpt EQU $0E0
104
105 bplcon0 EQU $100
106 bplcon1 EQU $102
107 bplcon2 EQU $104
108 bpllmod EQU $108
109 bpl2mod EQU $10A
110
111 bpldat EQU $110
112
113 sprpt EQU $120
114
115 spr EQU $140
116 * STRUCTURE SpriteDef
117 sd_pos EQU $00
118 sd_ctl EQU $02
119 sd_dataa EQU $04
120 sd_datab EQU $08
121
122 color EQU $180

```

```

1 *****
2 * Commodore-Amiga, Inc.
3 * dmabits.i
4 *
5 * $Header: dmabits.i,v 27.1 85/06/24 14:43:02 neil Exp $
6 *
7 * $Locker: $
8 *
9 *****
10
11 IFND HARDWARE_DMABITS_I
12 HARDWARE_DMABITS_I SET 1
13
14
15 * include file for defining dma control stuff */
16
17 * write definitions for dmaconw */
18 DMAF_SETCLR EQU $8000
19 DMAF_AUDIO EQU $000F /* 4 bit mask */
20 DMAF_AUD0 EQU $0001
21 DMAF_AUD1 EQU $0002
22 DMAF_AUD2 EQU $0004
23 DMAF_AUD3 EQU $0008
24 DMAF_DISK EQU $0010
25 DMAF_SPRITE EQU $0020
26 DMAF_BLITTER EQU $0040
27 DMAF_COPPER EQU $0080
28 DMAF_RASTER EQU $0100
29 DMAF_MASTER EQU $0200
30 DMAF_BLITHOG EQU $0400
31 DMAF_ALL EQU $01FF /* all dma channels */
32
33 * read definitions for dmaconr */
34 * bits 0-8 correspnd to dmaconw definitions */
35 DMAF_BLTDONE EQU $4000
36 DMAF_BLTNZERO EQU $2000
37
38 DMAB_SETCLR EQU 15
39 DMAB_AUD0 EQU 0
40 DMAB_AUD1 EQU 1
41 DMAB_AUD2 EQU 2
42 DMAB_AUD3 EQU 3
43 DMAB_DISK EQU 4
44 DMAB_SPRITE EQU 5
45 DMAB_BLITTER EQU 6
46 DMAB_COPPER EQU 7
47 DMAB_RASTER EQU 8
48 DMAB_MASTER EQU 9
49 DMAB_BLITHOG EQU 10
50 DMAB_BLTDONE EQU 14
51 DMAB_BLTNZERO EQU 13
52
53

```

```

1 *****
2 * Commodore-Amiga, Inc.
3 * intenabits.i -- definitions for the bits in the interrupt enable
4 * (and interrupt request) register
5 *
6 * $Header: intbits.i,v 27.1 85/06/24 14:43:07 neil Exp $
7 *
8 * $Locker: $
9 *
10 *****/
11
12 IFND HARDWARE_INTBITS_I
13 HARDWARE_INTBITS_I SET 1
14
15
16 INTB_SETCLR EQU (15) ;Set/Clear control bit. Determines if bits
17 ;written with a 1 get set or cleared. Bits
18 ;written with a zero are always unchanged.
19 INTB_INTEN EQU (14) ;Master interrupt (enable only)
20 INTB_EXTER EQU (13) ;External interrupt
21 INTB_DSKSYNC EQU (12) ;Disk re-SYNChronized
22 INTB_RBF EQU (11) ;serial port Receive Buffer Full
23 INTB_AUD3 EQU (10) ;Audio channel 3 block finished
24 INTB_AUD2 EQU (9) ;Audio channel 2 block finished
25 INTB_AUD1 EQU (8) ;Audio channel 1 block finished
26 INTB_AUD0 EQU (7) ;Audio channel 0 block finished
27 INTB_BLIT EQU (6) ;Blitter finished
28 INTB_VERTB EQU (5) ;start of Vertical Blank
29 INTB_COPER EQU (4) ;Coprocessor
30 INTB_PORTS EQU (3) ;I/O Ports and timers
31 INTB_SOFTINT EQU (2) ;software interrupt request
32 INTB_DSKBLK EQU (1) ;Disk Block done
33 INTB_TBE EQU (0) ;serial port Transmit Buffer Empty
34
35
36
37 INTF_SETCLR EQU (1<<15)
38 INTF_INTEN EQU (1<<14)
39 INTF_EXTER EQU (1<<13)
40 INTF_DSKSYNC EQU (1<<12)
41 INTF_RBF EQU (1<<11)
42 INTF_AUD3 EQU (1<<10)
43 INTF_AUD2 EQU (1<<9)
44 INTF_AUD1 EQU (1<<8)
45 INTF_AUD0 EQU (1<<7)
46 INTF_BLIT EQU (1<<6)
47 INTF_VERTB EQU (1<<5)
48 INTF_COPER EQU (1<<4)
49 INTF_PORTS EQU (1<<3)
50 INTF_SOFTINT EQU (1<<2)
51 INTF_DSKBLK EQU (1<<1)
52 INTF_TBE EQU (1<<0)
53
54

```

## Contents

```

/intuition/intuition.i
/intuition/intuitionbase.i

```

```

1:
2: IFND INTUITION_INTUITION_I
3: INTUITION_INTUITION_I SET 1
4:
5: ;** intuition.i ****
6: ;* Commodore-Amiga, Inc.
7: ;*
8: ;* intuition.i main include file for assembly-language
 programmers
9: ;*
10: ;* Modification History
11: ;* date : author : Comments
12: ;* -----
13: ;* 1-30-85 --RJ-- created this file!
14: ;* 6-12-85 Dale and Carl translated this from the
 c version
15: ;* 6-13-85 =VoodooDrJ= added back the comments
16: ;*
17: ;*****/
18:
19: IFND GRAPHICS_GFX_I
20: include 'graphics/gfx.i'
21: ENDC
22:
23: IFND GRAPHICS_CLIP_I
24: include 'graphics/clip.i'
25: ENDC
26:
27: IFND GRAPHICS_VIEW_I
28: include 'graphics/view.i'
29: ENDC
30:
31: IFND GRAPHICS_RASTPORT_I
32: include 'graphics/rastport.i'
33: ENDC
34:
35: IFND GRAPHICS_LAYERS_I
36: include 'graphics/layers.i'
37: ENDC
38:
39: IFND GRAPHICS_TEXT_I
40: include 'graphics/text.i'
41: ENDC
42:
43: IFND EXEC_PORTS_I
44: include 'exec/ports.i'
45: ENDC
46:
47: IFND DEVICES_TIMER_I
48: include 'devices/timer.i'
49: ENDC
50:
51: IFND DEVICES_INPTEVENT_I
52: include 'devices/inpthevent.i'
53: ENDC
54:
55: ; =====
56: ; === Menu =====
57: ; =====
58: ; =====
59: STRUCTURE Menu,0
60:
61: APTR mu_NextMenu ; menu pointer, same level
62: WORD mu_LeftEdge ; dimensions of the select box;
63: WORD mu_TopEdge ; dimensions of the select box;
64: WORD mu_Width ; dimensions of the select box;
65: WORD mu_Height ; dimensions of the select box;
66: WORD mu_Flags ; see flag definitions below;
67: APTR mu_MenuName ; text for this Menu header
68: APTR mu_FirstItem ; pointer to first in chain;
69:
70: ; these mysteriously-named variables are for internal
 use only
71: WORD mu_JazzX
72: WORD mu_JazzY
73: WORD mu_BeatX
74: WORD mu_BeatY
75:
76: LABEL mu_SIZEOF
77:
78: ; FLAGS SET BY BOTH THE APPLIPROG AND INTUITION
79: MENUENABLED equ $0001 ; whether or not this menu is enabled;
80:
81: ; FLAGS SET BY INTUITION;
82: MIDDRAWN equ $0100 ; this menu's items are currently drawn;
83:
84:
85:
86:
87:
88: ; =====
89: ; === MenuItem =====
90: ; =====
91: STRUCTURE MenuItem,0
92:
93: APTR mi_NextItem ; pointer to next in chained list
94: WORD mi_LeftEdge ; dimensions of the select box
95: WORD mi_TopEdge ; dimensions of the select box
96: WORD mi_Width ; dimensions of the select box
97: WORD mi_Height ; dimensions of the select box
98: WORD mi_Flags ; see the defines below
99:
100: LONG mi_MutualExclude ; set bits mean this item excludes
 that item
101:
102: APTR mi_ItemFill ; points to Image, IntuiText, or NULL
103:
104: ; when this item is pointed to by the cursor and the
 items highlight
105: ; mode HIGHIMAGE is selected, this alternate image will
 be displayed
106: APTR mi_SelectFill ; points to Image, IntuiText, or
 NULL
107:
108: BYTE mi_Command ; only if appliprogram sets the COMMSEQ
 flag
109:
110: ; The following variable is strictly from Kludge-City,
 where some people

```

```

111: ; still live. It is included solely because our types.i
 macros aren't
112: ; smart enough to do the right thing, which would be
 the automatic
113: ; word-alignment to these references as it SHOULD be
 in order to duplicate
114: ; the way alignments are adjusted in the c-language.
 And instead of
115: ; correcting the problem, I am obliged to kludge up
 my include.i files.
116: ; So here it is!
117: BYTE mi_KludgeFill00 ; defined as a BYTE because this
 does
118:
119: APTR mi_SubItem ; if non-zero, DrawMenu shows "->"
120:
121: ; The NextSelect field represents the menu number of
 next selected
122: ; item (when user has drag-selected several items)
123: WORD mi_NextSelect
124:
125: LABEL mi_SIZEOF
126:
127: ; --- FLAGS SET BY THE APPLIPROG -----
128: CHECKIT equ $0001 ; whether to check this item if
 selected
129: ITEMTEXT equ $0002 ; set if textual, clear if graphical
 item
130: COMMSEQ equ $0004 ; set if there's an command sequence
131: MENUTOGGLE equ $0008 ; set to toggle the check of a menu
 item
132: ITEMENABLED equ $0010 ; set if this item is enabled
133:
134: ; these are the SPECIAL HIGHLIGHT FLAG state meanings
135: HIGHFLAGS equ $00C0 ; see definitions below for these
 bits
136: HIGHIMAGE equ $0000 ; use the user's "select image"
137: HIGHCOMP equ $0040 ; highlight by complementing the select
 box
138: HIGHBOX equ $0080 ; highlight by drawing a box around
 the image
139: HIGHNONE equ $00C0 ; don't highlight
140:
141: ; --- FLAGS SET BY BOTH APPLIPROG AND INTUITION -----
142: CHECKED equ $0100 ; if CHECKIT, then set this when selected
143:
144:
145: ; --- FLAGS SET BY INTUITION -----
146: ISDRAWN equ $1000 ; this item's subs are currently
 drawn
147: HIGHITEM equ $2000 ; this item is currently highlighted
148: MENUTOGGLED equ $4000 ; this item was already toggled
149:
150:
151:
152:
153:
154:
155: ; =====

156: ; === Requester =====
157: ; =====
158: STRUCTURE Requester,0
159:
160: ; the ClipRect and BitMap and used for rendering the
 requester
161: APTR rq_OlderRequest
162: WORD rq_LeftEdge ; dimensions of the entire box
163: WORD rq_TopEdge ; dimensions of the entire box
164: WORD rq_Width ; dimensions of the entire box
165: WORD rq_Height ; dimensions of the entire box
166:
167: WORD rq_RelLeft ; get POINTREL Pointer relativity
 offsets
168: WORD rq_RelTop ; get POINTREL Pointer relativity
 offsets
169:
170: APTR rq_ReqGadget ; pointer to the first of
 a list of gadgets
171: APTR rq_ReqBorder ; the box's border
172: APTR rq_ReqText ; the box's text
173:
174: USHORT rq_Flags ; see definitions below
175:
176: UBYTE rq_BackFill ; pen number for back-plane fill
 before draws
177:
178: ; The following variable is strictly from Kludge-City,
 where some people
179: ; still live. It is included solely because our types.i
 macros aren't
180: ; smart enough to do the right thing, which would be
 the automatic
181: ; word-alignment to these references as it SHOULD be
 in order to duplicate
182: ; the way alignments are adjusted in the c-language.
 And instead of
183: ; correcting the problem, I am obliged to kludge up
 my include.i files.
184: ; So here it is!
185: BYTE rq_KludgeFill00 ; defined as a BYTE because this
 does
186:
187: APTR rq_ReqLayer ; layer in which requester rendered
188: STRUCT rq_ReqPad1,32 ; for backwards compatibility
 (reserved)
189:
190: ; If the BitMap plane pointers are non-zero, this tells
 the system
191: ; that the image comes pre-drawn (if the appliprogram wants
 to define
192: ; it's own box, in any shape or size it wants!); this
 is OK by
193: ; Intuition as long as there's a good correspondence
 between the image
194: ; and the specified Gadgets
195: APTR rq_ReqBMap ; points to the BitMap of PREDRAWN
 imagery
196:
197: APTR rq_RWindow ; points back to requester's window

```

```

198: STRUCT rq_ReqPad2,36 ; for backwards compatibility
 (reserved)
199:
200: LABEL rq_SIZEOF
201:
202: ; FLAGS SET BY THE APPLIPROG
203: POINTREL equ $0001 ; if POINTREL set, TopLeft is relative
 to pointer
204: PREDRAWN equ $0002 ; if ReqBMap points to predrawn Requester
 imagery
205:
206: ; FLAGS SET BY INTUITION;
207: REQOFFWINDOW equ $1000 ; part of one of the Gadgets
 was offwindow
208: REQACTIVE equ $2000 ; this requester is active
209: SYSREQUEST equ $4000 ; this requester caused by system
210: DEFERREFRESH equ $8000 ; this Requester stops a Refresh
 broadcast
211:
212:
213:
214:
215:
216: ; =====
217: ; --- Gadget -----
218: ; =====
219: STRUCTURE Gadget,0
220:
221: APTR gg_NextGadget ; next gadget in the list
222:
223: WORD gg_LeftEdge ; "hit box" of gadget
224: WORD gg_TopEdge ; "hit box" of gadget
225: WORD gg_Width ; "hit box" of gadget
226: WORD gg_Height ; "hit box" of gadget
227:
228: WORD gg_Flags ; see below for list of defines
229:
230: WORD gg_Activation ; see below for list of defines
231:
232: WORD gg_GadgetType ; see below for defines
233:
234: ; appliprogram can specify that the Gadget be rendered
 as either as Border
235: ; or an Image. This variable points to which (or equals
 NULL if there's
236: ; nothing to be rendered about this Gadget)
237: APTR gg_GadgetRender
238:
239: ; appliprogram can specify "highlighted" imagery rather
 than algorithmic
240: ; this can point to either Border or Image data
241: APTR gg_SelectRender
242:
243: APTR gg_GadgetText ; text for this gadget;
244:
245: ; by using the MutualExclude word, the appliprogram can
 describe
246: ; which gadgets mutually-exclude which other ones.
 The bits in
247: ; MutualExclude correspond to the gadgets in object
 containing
248: ; the gadget list. If this gadget is selected and a
 bit is set
249: ; in this gadget's MutualExclude and the gadget corresponding
 to
250: ; that bit is currently selected (e.g. bit 2 set and
 gadget 2
251: ; is currently selected) that gadget must be unselected.
 Intuition
252: ; does the visual unselecting (with checkmarks) and
 leaves it up
253: ; to the program to unselect internally
254: LONG gg_MutualExclude ; set bits mean this gadget excludes
 that
255:
256: ; pointer to a structure of special data required by
 Proportional, String
257: ; and Integer Gadgets
258: APTR gg_SpecialInfo
259:
260: WORD gg_GadgetID ; user-definable ID field
261: APTR gg_UserData ; ptr to general purpose User data
 (ignored by Intuit)
262:
263: LABEL gg_SIZEOF
264:
265: ; --- FLAGS SET BY THE APPLIPROG -----
266: ; combinations in these bits describe the highlight technique
 to be used
267: GADGHIGHBITS equ $0003
268: GADGHCOMP equ $0000 ; Complement the select box
269: GADGHBOX equ $0001 ; Draw a box around the image
270: GADGHIMAGE equ $0002 ; Blast in this alternate image
271: GADGHNONE equ $0003 ; don't highlight
272:
273: ; set this flag if the GadgetRender and SelectRender point
 to Image imagery,
274: ; clear if it's a Border
275: GADGIMAGE equ $0004
276:
277: ; combinations in these next two bits specify to which corner
 the gadget's
278: ; Left & Top coordinates are relative. If relative to Top/Left,
279: ; these are "normal" coordinates (everything is relative
 to something in
280: ; this universe)
281: GRELBOTTOM equ $0008 ; set if rel to bottom, clear if
 rel top
282: GRELRIGHT equ $0010 ; set if rel to right, clear if
 to left
283: ; set the RELWIDTH bit to spec that Width is relative to
 width of screen
284: GRELWIDTH equ $0020
285: ; set the RELHEIGHT bit to spec that Height is rel to height
 of screen
286: GRELHEIGHT equ $0040
287:
288: ; the SELECTED flag is initialized by you and set by Intuition.
 It
289: ; specifies whether or not this Gadget is currently selected/highlighted

```

```

290: SELECTED equ $0080
291:
292:
293: ; the GADGDISABLED flag is initialized by you and later
 set by Intuition
294: ; according to your calls to On/OffGadget(). It specifies
 whether or not
295: ; this Gadget is currently disabled from being selected
296: GADGDISABLED equ $0100
297:
298:
299: ; --- These are the Activation flag bits -----
300: ; RELVERIFY is set if you want to verify that the pointer
 was still over
301: ; the gadget when the select button was released
302: RELVERIFY equ $0001
303:
304: ; the flag GADGIMMEDIATE, when set, informs the caller that
 the gadget
305: ; was activated when it was activated. this flag works
 in conjunction with
306: ; the RELVERIFY flag
307: GADGIMMEDIATE equ $0002
308:
309: ; the flag ENDGADGET, when set, tells the system that this
 gadget, when
310: ; selected, causes the Requester or AbsMessage to be ended.
 Requesters or
311: ; AbsMessages that are ended are erased and unlinked from
 the system
312: ENDGADGET equ $0004
313:
314: ; the FOLLOWMOUSE flag, when set, specifies that you want
 to receive
315: ; reports on mouse movements (ie, you want the REPORTMOUSE
 function for
316: ; your Window). When the Gadget is deselected (immediately
 if you have
317: ; no RELVERIFY) the previous state of the REPORTMOUSE flag
 is restored
318: ; You probably want to set the GADGIMMEDIATE flag when using
 FOLLOWMOUSE,
319: ; since that's the only reasonable way you have of learning
 why Intuition
320: ; is suddenly sending you a stream of mouse movement events.
 If you don't
321: ; set RELVERIFY, you'll get at least one Mouse Position
 event.
322: FOLLOWMOUSE equ $0008
323:
324: ; if any of the BORDER flags are set in a Gadget that's
 included in the
325: ; Gadget list when a Window is opened, the corresponding
 Border will
326: ; be adjusted to make room for the Gadget
327: RIGHTBORDER equ $0010
328: LEFTBORDER equ $0020
329: TOPBORDER equ $0040
330: BOTTBORDER equ $0080
331:

```

```

332: TOGGLESELECT equ $0100 ; this bit for toggle-select
 mode
333:
334: STRINGCENTER equ $0200 ; center the String
335: STRINGRIGHT equ $0400 ; right-justify the String
336:
337: LONGINT equ $0800 ; This String Gadget is a Long Integer
338:
339: ALTKEYMAP equ $1000 ; This String has an alternate keymapping
340:
341:
342: ; --- GADGET TYPES -----
343: ; These are the Gaget Type definitions for the variable
 GadgetType.
344: ; Gadget number type MUST start from one. NO TYPES OF ZERO
 ALLOWED.
345: ; first comes the mask for Gadget flags reserved for Gadget
 typing
346: GADGETTYPE equ $FC00 ; all Gadget Global Type flags (padded)
347: SYSGADGET equ $8000 ; 1 = SysGadget, 0 = AppliGadget
348: SCRGADGET equ $4000 ; 1 = ScreenGadget, 0 = WindowGadget
349: GZZGADGET equ $2000 ; 1 = Gadget for GIMMEZEROZERO borders
350: REQGADGET equ $1000 ; 1 = this is a Requester Gadget
351: ; system gadgets
352: SIZING equ $0010
353: WDRAGGING equ $0020
354: SDRAGGING equ $0030
355: WUPFRONT equ $0040
356: SUPFRONT equ $0050
357: WDOWNBACK equ $0060
358: SDOWNBACK equ $0070
359: CLOSE equ $0080
360: ; application gadgets
361: BOOLGADGET equ $0001
362: GADGET0002 equ $0002
363: PROPGADGET equ $0003
364: STRGADGET equ $0004
365:
366:
367:
368:
369:
370:
371: ; =====
372: ; == PropInfo =====
373: ; =====
374: ; this is the special data required by the proportional
 Gadget
375: ; typically, this data will be pointed to by the Gadget
 variable SpecialInfo
376: STRUCTURE PropInfo,0
377:
378: WORD pi_Flags ; general purpose flag bits (see defines
 below)
379:
380: ; You initialize the Pot variables before the Gadget
 is added to
381: ; the system. Then you can look here for the current
 settings
382: ; any time, even while User is playing with this Gadget.

```

```

383: To
 ; adjust these after the Gadget is added to the System,
 use
384: ; ModifyProp(); The Pots are the actual proportional
 settings,
385: ; where a value of zero means zero and a value of MAXPOT
 means
386: ; that the Gadget is set to its maximum setting.
387: WORD pi_HorizPot ; 16-bit FixedPoint horizontal quantity
 percentage;
388: WORD pi_VertPot ; 16-bit FixedPoint vertical quantity
 percentage;
389:
390: ; the 16-bit FixedPoint Body variables describe what
 percentage
391: ; of the entire body of stuff referred to by this Gadget
 is
392: ; actually shown at one time. This is used with the
 AUTOKNOB
393: ; routines, to adjust the size of the AUTOKNOB according
 to how
394: ; much of the data can be seen. This is also used to
 decide how
395: ; far to advance the Pots when User hits the Container
 of the Gadget.
396: ; For instance, if you were controlling the display
 of a 5-line
397: ; Window of text with this Gadget, and there was a total
 of 15
398: ; lines that could be displayed, you would set the VertBody
 value to
399: ; (MAXBODY / (TotalLines / DisplayLines)) = MAXBODY
 / 3.
400: ; Therefore, the AUTOKNOB would fill 1/3 of the container,
 and if
401: ; User hits the Cotainer outside of the knob, the pot
 would advance
402: ; 1/3 (plus or minus) If there's no body to show, or
 the total
403: ; amount of displayable info is less than the display
 area, set the
404: ; Body variables to the MAX. To adjust these after
 the Gadget is
405: ; added to the System, use ModifyProp().
406: WORD pi_HorizBody ; horizontal Body
407: WORD pi_VertBody ; vertical Body
408:
409: ; these are the variables that Intuition sets and maintains
410: WORD pi_CWidth ; Container width (with any relativity
 absolved)
411: WORD pi_CHeight ; Container height (with any relativity
 absolved)
412: WORD pi_HPOTRes ; pot increments
413: WORD pi_VPotRes ; pot increments
414: WORD pi_LeftBorder ; Container borders
415: WORD pi_TopBorder ; Container borders
416: LABEL pi_SIZEOF
417:
418: ; --- FLAG BITS -----
419: AUTOKNOB equ $0001 ; this flag sez: gimme that old auto-knob
420: FREEHORIZ equ $0002 ; if set, the knob can move horizontally
421: FREEVERT equ $0004 ; if set, the knob can move vertically
422: PROPBORDERLESS equ $0008 ; if set, no border will be rendered
423: KNOBHIT equ $0100 ; set when this Knob is hit
424:
425:
426: KNOBHMIn equ 6 ; minimum horizontal size of the knob
427: KNOBVMIN equ 4 ; minimum vertical size of the knob
428: MAXBODY equ $FFFF ; maximum body value
429: MAXPOT equ $FFFF ; maximum pot value
430:
431:
432:
433:
434:
435:
436: ; =====
437: ; --- StringInfo -----
438: ; =====
439: ; this is the special data required by the string Gadget
440: ; typically, this data will be pointed to by the Gadget
 variable SpecialInfo
441: STRUCTURE StringInfo,0
442:
443: ; you initialize these variables, and then Intuition
 maintains them
444: APTR si_Buffer ; the buffer containing the start and
 final string
445: APTR si_UndoBuffer ; optional buffer for undoing current
 entry
446: WORD si_BufferPos ; character position in Buffer
447: WORD si_MaxChars ; max number of chars in Buffer (including
 NULL)
448: WORD si_DisPPos ; Buffer position of first displayed
 character
449:
450: ; Intuition initializes and maintains these variables
 for you
451: WORD si_UndoPos ; character position in the undo buffer
452: WORD si_NumChars ; number of characters currently in
 Buffer
453: WORD si_DisPCount ; number of whole characters visible
 in Container
454: WORD si_CLeft ; topleft offset of the container
455: WORD si_CTop ; topleft offset of the container
456: APTR si_LayerPtr ; the RastPort containing this Gadget
457:
458: ; you can initialize this variable before the gadget
 is submitted to
459: ; Intuition, and then examine it later to discover what
 integer
460: ; the user has entered (if the user never plays with
 the gadget,
461: ; the value will be unchanged from your initial setting)
462: LONG si_LongInt ; the LONG return value of a LONGINT
 String Gadget
463:
464: ; If you want this Gadget to use your own Console keymapping,
 you
465: ; set the ALTKEYMAP bit in the Activation flags of the

```



```

466: Gadget, and then
 ; set this variable to point to your keymap. If you
 don't set the
467: ; ALTKEYMAP, you'll get the standard ASCII keymapping.
468: APTR si_AltKeyMap
469:
470: LABEL si_SIZEOF
471:
472:
473:
474:
475: ; =====
476: ; === IntuiText =====
477: ; =====
478: ; IntuiText is a series of strings that start with a screen
 location
479: ; (always relative to the upper-left corner of something)
 and then the
480: ; text of the string. The text is null-terminated.
481: STRUCTURE IntuiText,0
482:
483: UBYTE it_FrontPen ; the pens for rendering the
 text
484: UBYTE it_BackPen ; the pens for rendering the text
485:
486: UBYTE it_DrawMode ; the mode for rendering the
 text
487:
488: ; The following variable is strictly from Kludge-City,
 where some people
489: ; still live. It is included solely because our types.i
 macros aren't
490: ; smart enough to do the right thing, which would be
 the automatic
491: ; word-alignment to these references as it SHOULD be
 in order to duplicate
492: ; the way alignments are adjusted in the c-language.
 And instead of
493: ; correcting the problem, I am obliged to kludge up
 my include.i files.
494: ; So here it is!
495: BYTE it_KludgeFill100 ; defined as a BYTE because this
 does
496:
497: WORD it_LeftEdge ; relative start location for the
 text
498: WORD it_TopEdge ; relative start location for the
 text
499:
500: APTR it_ITextFont ; if NULL, you accept the defaults
501:
502: APTR it_IText ; pointer to null-terminated text
503:
504: APTR it_NextText ; continuation to TxWrite another
 text
505:
506: LABEL it_SIZEOF
507:
508:
509:
510:
511:
512: ; =====
513: ; === Border =====
514: ; =====
515: ; Data type Border, used for drawing a series of lines which
 is intended for
516: ; use as a border drawing, but which may, in fact, be used
 to render any
517: ; arbitrary vector shape.
518: ; The routine DrawBorder sets up the RastPort with the appropriate
519: ; variables, then does a Move to the first coordinate, then
 does Draws
520: ; to the subsequent coordinates.
521: ; After all the Draws are done, if NextBorder is non-zero
 we call DrawBorder
522: ; recursively
523: STRUCTURE Border,0
524:
525: WORD bd_LeftEdge ; initial offsets from the origin
526: WORD bd_TopEdge ; initial offsets from the origin
527: UBYTE bd_FrontPen ; pen number for rendering
528: UBYTE bd_BackPen ; pen number for rendering
529: UBYTE bd_DrawMode ; mode for rendering
530: BYTE bd_Count ; number of XY pairs
531: APTR bd_XY ; vector coordinate pairs rel to
 LeftTop
532: APTR bd_NextBorder ; pointer to any other Border
 too
533:
534: LABEL bd_SIZEOF
535:
536:
537: ; =====
538: ; === Image =====
539: ; =====
540: ; This is a brief image structure for very simple transfers
 of
541: ; image data to a RastPort
542: STRUCTURE Image,0
543:
544: WORD ig_LeftEdge ; starting offset relative to something
545:
546: WORD ig_TopEdge ; starting offset relative to something
547:
548: WORD ig_Width ; pixel size (though data is word-aligned)
549: WORD ig_Height ; pixel size
550: WORD ig_Depth ; pixel size
551: APTR ig_ImageData ; pointer to the actual image
 bits
552:
553: ; the PlanePick and PlaneOnOff variables work much the
 same way as the
554: ; equivalent GELS Bob variables. It's a space-saving
555: ; mechanism for image data. Rather than defining the
 image data

```

```

554: ; for every plane of the RastPort, you need define data
 only for planes
555: ; that are not entirely zero or one. As you define
 your Imagery, you will
556: ; often find that most of the planes ARE just as color
 selectors. For
557: ; instance, if you're designing a two-color Gadget to
 use colors two and
558: ; three, and the Gadget will reside in a five-plane
 display, plane zero
559: ; of your imagery would be all ones, bit plane one would
 have data that
560: ; describes the imagery, and bit planes two through
 four would be
561: ; all zeroes. Using these flags allows you to avoid
 wasting all that
562: ; memory in this way:
563: ; first, you specify which planes you want your data
 to appear
564: ; in using the PlanePick variable. For each bit set
 in the variable, the
565: ; next "plane" of your image data is blitted to the
 display. For each bit
566: ; clear in this variable, the corresponding bit in PlaneOnOff
 is examined.
567: ; If that bit is clear, a "plane" of zeroes will be
 used. If the bit is
568: ; set, ones will go out instead. So, for our example:
569: ; Gadget.PlanePick = 0x02;
570: ; Gadget.PlaneOnOff = 0x01;
571: ; Note that this also allows for generic Gadgets, like
 the System Gadgets,
572: ; which will work in any number of bit planes
573: ; Note also that if you want an Image that is only a
 filled rectangle,
574: ; you can get this by setting PlanePick to zero (pick
 no planes of data)
575: ; and set PlaneOnOff to describe the pen color of the
 rectangle.
576: BYTE ig_PlanePick
577: BYTE ig_PlaneOnOff
578:
579: ; if the NextImage variable is not NULL, Intuition presumes
 that
580: ; it points to another Image structure with another
 Image to be
581: ; rendered
582: APTR ig_NextImage
583:
584:
585: LABEL ig_SIZEOF
586:
587:
588:
589:
590: ; =====
591: ; === IntuiMessage =====
592: ; =====

```

```

593: STRUCTURE IntuiMessage,0
594:
595: STRUCT im_ExecMessage,MN_SIZE
596:
597: ; the Class bits correspond directly with the IDCMP
 Flags, except for the
598: ; special bit LONELYMESSAGE (defined below)
599: ULONG im_Class
600:
601: ; the Code field is for special values like MENU number
602: WORD im_Code
603:
604: ; the Qualifier field is a copy of the current InputEvent's
 Qualifier
605: WORD im_Qualifier
606:
607: ; IAddress contains particular addresses for Intuition
 functions, like
608: ; the pointer to the Gadget or the Screen
609: APTR im_IAddress
610:
611: ; when getting mouse movement reports, any event you
 get will have the
612: ; the mouse coordinates in these variables. the coordinates
 are relative
613: ; to the upper-left corner of your Window (GIMMEZEROZERO
 notwithstanding)
614: WORD im_MouseX
615: WORD im_MouseY
616:
617: ; the time values are copies of the current system clock
 time. Micros
618: ; are in units of microseconds, Seconds in seconds.
619: LONG im_Seconds
620: LONG im_Micros
621:
622: ; the IDCMPWindow variable will always have the address
 of the Window of
623: ; this IDCMP
624: APTR im_IDCMPWindow
625:
626: ; system-use variable
627: APTR im_SpecialLink
628:
629: LABEL im_SIZEOF
630:
631:
632:
633: ; --- IDCMP Classes -----
634: SIZEVERIFY equ $00000001 ; See the Programmer's Guide
635: NEWSIZE equ $00000002 ; See the Programmer's Guide
636: REFRESHWINDOW equ $00000004 ; See the Programmer's
 Guide
637: MOUSEBUTTONS equ $00000008 ; See the Programmer's
 Guide
638: MOUSEMOVE equ $00000010 ; See the Programmer's Guide

```

```

639: GADGETDOWN equ $00000020 ; See the Programmer's Guide
640: GADGETUP equ $00000040 ; See the Programmer's Guide

641: REQSET equ $00000080 ; See the Programmer's Guide
642: MENUPICK equ $00000100 ; See the Programmer's Guide

643: CLOSEWINDOW equ $00000200 ; See the Programmer's Guide
644: RAWKEY equ $00000400 ; See the Programmer's Guide

645: REQVERIFY equ $00000800 ; See the Programmer's Guide
646: REQCLEAR equ $00001000 ; See the Programmer's Guide

647: MENUVERIFY equ $00002000 ; See the Programmer's Guide
648: NEWPREFS equ $00004000 ; See the Programmer's Guide

649: DISKINSERTED equ $00008000 ; See the Programmer's
 Guide
650: DISKREMOVED equ $00010000 ; See the Programmer's Guide

651: WBENCHMESSAGE equ $00020000 ; See the Programmer's
 Guide
652: ACTIVEWINDOW equ $00040000 ; See the Programmer's
 Guide
653: INACTIVEWINDOW equ $00080000 ; See the Programmer's
 Guide
654: DELTAMOVE equ $00100000 ; See the Programmer's Guide
655: VANILLAKEY equ $00200000 ; See the Programmer's Guide
656: INTUITICKS equ $00400000 ; See the Programmer's Guide
657: ; NOTEZ-BIEN: $80000000 is reserved for internal use
 by IDCMP

658:
659: ; the IDCMP Flags do not use this special bit, which is
 cleared when
660: ; Intuition sends its special message to the Task, and set
 when Intuition
661: ; gets its Message back from the Task. Therefore, I can
 check here to
662: ; find out fast whether or not this Message is available
 for me to send
663: LONELYMESSAGE equ $80000000
664:
665:
666:
667: ; --- IDCMP Codes -----

668: ; This group of codes is for the MENUVERIFY function
669: MENUHOT equ $0001 ; IntuiWants verification or MENUCANCEL

670: MENUCANCEL equ $0002 ; HOT Reply of this cancels Menu
 operation
671: MENUWAITING equ $0003 ; Intuition simply wants a ReplyMsg()
 ASAP

672:
673: ; This group of codes is for the WBENCHMESSAGE messages
674: WBENCHOPEN equ $0001
675: WBENCHCLOSE equ $0002
676:
677:
678:
679:

```

```

680: ; =====
681: ; === Window =====
682: ; =====

683: STRUCTURE Window,0
684:
685: APTR wd_NextWindow ; for the linked list of a Screen
686:
687: WORD wd_LeftEdge ; screen dimensions
688: WORD wd_TopEdge ; screen dimensions
689: WORD wd_Width ; screen dimensions
690: WORD wd_Height ; screen dimensions
691:
692: WORD wd_MouseY ; relative top top-left corner
693: WORD wd_MouseX ; relative top top-left corner
694:
695: WORD wd_MinWidth ; minimum sizes
696: WORD wd_MinHeight ; minimum sizes
697: WORD wd_MaxWidth ; maximum sizes
698: WORD wd_MaxHeight ; maximum sizes
699:
700: LONG wd_Flags ; see below for definitions
701:
702: APTR wd_MenuStrip ; first in a list of menu headers
703:
704: APTR wd_Title ; title text for the Window
705:
706: APTR wd_FirstRequest ; first in linked list of active
 Requesters
707: APTR wd_DMRequest ; the double-menu Requester
708: WORD wd_ReqCount ; number of Requesters blocking
 this Window
709: APTR wd_WScreen ; this Window's Screen
710: APTR wd_RPort ; this Window's very own RastPort
711:
712: ; the border variables describe the window border.
 If you specify
713: ; GIMMEZEROZERO when you open the window, then the upper-left
 of the
714: ; ClipRect for this window will be upper-left of the
 BitMap (with correct
715: ; offsets when in SuperBitMap mode; you MUST select
 GIMMEZEROZERO when
716: ; using SuperBitMap). If you don't specify ZeroZero,
 then you save
717: ; memory (no allocation of RastPort, Layer, ClipRect
 and associated
718: ; Bitmaps), but you also must offset all your writes
 by BorderTop,
719: ; BorderLeft and do your own mini-clipping to prevent
 writing over the
720: ; system gadgets
721: BYTE wd_BorderLeft
722: BYTE wd_BorderTop
723: BYTE wd_BorderRight
724: BYTE wd_BorderBottom
725: APTR wd_BorderRPort
726:

```

```

727: ; You supply a linked-list of gadget that you want for
your Window.
728: ; This list DOES NOT include system Gadgets. You get
the standard
729: ; window system Gadgets by setting flag-bits in the
variable Flags (see
730: ; the bit definitions below)
731: APTR wd_FirstGadget
732:
733: ; these are for opening/closing the windows
734: APTR wd_Parent
735: APTR wd_Descendant
736:
737: ; sprite data information for your own Pointer
738: ; set these AFTER you Open the Window by calling SetPointer()
739: APTR wd_Pointer
740: BYTE wd_PtrHeight
741: BYTE wd_PtrWidth
742: BYTE wd_XOffset
743: BYTE wd_YOffset
744:
745: ; the IDCMP Flags and User's and Intuition's Message
Ports
746: ULONG wd_IDCMPFlags
747: APTR wd_UserPort
748: APTR wd_WindowPort
749: APTR wd_MessageKey
750:
751: BYTE wd_DetailPen
752: BYTE wd_BlockPen
753:
754: ; the CheckMark is a pointer to the imagery that will
be used when
755: ; rendering MenuItems of this Window that want to be
checkmarked
756: ; if this is equal to NULL, you'll get the default imagery
757: APTR wd_CheckMark
758:
759: ; if non-null, Screen title when Window is active
760: APTR wd_ScreenTitle
761:
762: ; These variables have the mouse coordinates relative
to the
763: ; inner-Window of GIMMEZEROZERO Windows. This is compared
with the
764: ; MouseX and MouseY variables, which contain the mouse
coordinates
765: ; relative to the upper-left corner of the Window, GIMMEZEROZERO
766: ; notwithstanding
767: SHORT wd_GZZMouseX
768: SHORT wd_GZZMouseY
769: ; these variables contain the width and height of the
inner-Window of
770: ; GIMMEZEROZERO Windows
771: SHORT wd_GZZWidth
772: SHORT wd_GZZHeight
773:
774: APTR wd_ExtData
775:
776: ; general-purpose pointer to User data extension

777: APTR wd_UserData
778: APTR wd_WLayer ; stash of Window.RPort->Layer
779:
780: LABEL wd_Size
781:
782: ; --- FLAGS REQUESTED (NOT DIRECTLY SET THOUGH) BY THE APPLIPROG

783: WINDOWSMIZING equ $0001 ; include sizing system-gadget?
784: WINDOWDRAG equ $0002 ; include dragging system-gadget?
785: WINDOWDEPTH equ $0004 ; include depth arrangement gadget?
786: WINDOWCLOSE equ $0008 ; include close-box system-gadget?
787:
788: SIZEBRIGHT equ $0010 ; size gadget uses right border
789: SIZEBBOTTOM equ $0020 ; size gadget uses bottom border
790:
791: ; --- refresh modes -----
792: ; combinations of the REFRESHBITS select the refresh type

793: REFRESHBITS equ $00C0
794: SMART_REFRESH equ $0000
795: SIMPLE_REFRESH equ $0040
796: SUPER_BITMAP equ $0080
797: OTHER_REFRESH equ $00C0
798:
799: BACKDROP equ $0100 ; this is an ever-popular BACKDROP
window
800:
801: REPORTMOUSE equ $0200 ; set this to hear about every mouse
move
802:
803: GIMMEZEROZERO equ $0400 ; make extra border stuff
804:
805: BORDERLESS equ $0800 ; set this to get a Window sans
border
806:
807: ACTIVATE equ $1000 ; when Window opens, it's the Active
one
808:
809: ; FLAGS SET BY INTUITION
810: WINDOWACTIVE equ $2000 ; this window is the active one
811:
811: INREQUEST equ $4000 ; this window is in request mode
812:
812: MENUSTATE equ $8000 ; this Window is active with
its Menus on
813:
814: ; --- Other User Flags -----
815: RMBTRAP equ $00010000 ; Catch RMB events for your own
816:
816: NOCAREREFRESH equ $00020000 ; not to be bothered with
REFRESH
817:
818:
819: WINDOWREFRESH equ $01000000 ; Window is currently refreshing

```

```

820: WBENCHWINDOW equ $02000000 ; WorkBench Window
821: WINDOWTICKED equ $04000000 ; only one timer tick at a
 time
822:
823: SUPER_UNUSED equ $FCFC0000 ;bits of Flag unused yet
824:
825:
826:
827:
828: ; --- see struct IntuiMessage for the IDCMP Flag definitions

829:
830:
831:
832:
833:
834:
835:
836:
837:
838: ; =====
839: ; === NewWindow =====
840: ; =====
841: STRUCTURE NewWindow,0
842:
843: WORD nw_LeftEdge ; initial Window dimensions
844: WORD nw_TopEdge ; initial Window dimensions
845: WORD nw_Width ; initial Window dimensions
846: WORD nw_Height ; initial Window dimensions
847:
848: BYTE nw_DetailPen ; for rendering the detail bits
 of the Window
849: BYTE nw_BlockPen ; for rendering the block-fill bits
850:
851: ULONG nw_IDCMPFlags ; initial IDCMP state
852:
853: LONG nw_Flags ; see the Flag definition under Window
854:
855: ; You supply a linked-list of Gadgets for your Window.
856: ; This list DOES NOT include system Gadgets. You get
 the standard
857: ; system Window Gadgets by setting flag-bits in the
 variable Flags (see
858: ; the bit definitions under the Window structure definition)
859: APTR nw_FirstGadget
860:
861: ; the CheckMark is a pointer to the imagery that will
 be used when
862: ; rendering MenuItems of this Window that want to be
 checkmarked
863: ; if this is equal to NULL, you'll get the default imagery
864: APTR nw_CheckMark
865:
866: APTR nw_Title ; title text for the Window
867:
868: ; the Screen pointer is used only if you've defined
 a CUSTOMSCREEN and
869: ; want this Window to open in it. If so, you pass the
 address of the
870: ; Custom Screen structure in this variable. Otherwise,
 this variable
871: ; is ignored and doesn't have to be initialized.
872: APTR nw_Screen
873:
874: ; SUPER_BITMAP Window? If so, put the address of your
 BitMap structure
875: ; in this variable. If not, this variable is ignored
 and doesn't have
876: ; to be initialized
877: APTR nw_BitMap
878:
879: ; the values describe the minimum and maximum sizes
 of your Windows.
880: ; these matter only if you've chosen the WINDOWSIZING
 Gadget option,
881: ; which means that you want to let the User to change
 the size of
882: ; this Window. You describe the minimum and maximum
 sizes that the
883: ; Window can grow by setting these variables. You can
 initialize
884: ; any one these to zero, which will mean that you want
 to duplicate
885: ; the setting for that dimension (if MinWidth == 0,
 MinWidth will be
886: ; set to the opening Width of the Window).
887: ; You can change these settings later using SetWindowLimits().
888: ; If you haven't asked for a SIZING Gadget, you don't
 have to
889: ; initialize any of these variables.
890: WORD nw_MinWidth
891: WORD nw_MinHeight
892: WORD nw_MaxWidth
893: WORD nw_MaxHeight
894:
895: ; the type variable describes the Screen in which you
 want this Window to
896: ; open. The type value can either be CUSTOMSCREEN or
 one of the
897: ; system standard Screen Types such as WBENCHSCREEN.
 See the
898: ; type definitions under the Screen structure
899: WORD nw_Type
900:
901: LABEL nw_SIZE
902:
903:
904:
905:
906: ; =====
907: ; === Screen =====
908: ; =====
909: STRUCTURE Screen,0

```

```

910:
911: APTR sc_NextScreen ; linked list of screens
912: APTR sc_FirstWindow ; linked list Screen's Windows
913:
914: WORD sc_LeftEdge ; parameters of the screen
915: WORD sc_TopEdge ; parameters of the screen
916:
917: WORD sc_Width ; null-terminated Title text
918: WORD sc_Height ; for Windows without ScreenTitle
919:
920: WORD sc_MouseY ; position relative to upper-left
921: WORD sc_MouseX ; position relative to upper-left
922:
923: WORD sc_Flags ; see definitions below
924:
925: APTR sc_Title
926: APTR sc_DefaultTitle
927:
928: ; Bar sizes for this Screen and all Window's in this
Screen
929: BYTE sc_BarHeight
930: BYTE sc_BarVBorder
931: BYTE sc_BarHBorder
932: BYTE sc_MenuVBorder
933: BYTE sc_MenuHBorder
934: BYTE sc_WBorTop
935: BYTE sc_WBorLeft
936: BYTE sc_WBorRight
937: BYTE sc_WBorBottom
938:
939: ; The following variable is strictly from Kludge-City,
where some people
940: ; still live. It is included solely because our types.i
macros aren't
941: ; smart enough to do the right thing, which would be
the automatic
942: ; word-alignment to these references as it SHOULD be
in order to duplicate
943: ; the way alignments are adjusted in the c-language.
And instead of
944: ; correcting the problem, I am obliged to kludge up
my include.i files.
945: ; So here it is!
946: BYTE sc_KludgeFill100 ; defined as a BYTE because this
does
947:
948: ; the display data structures for this Screen
949: APTR sc_Font ; this screen's default font
950: STRUCT sc_ViewPort, vp_SIZEOF ; describing the Screen's
display
951: STRUCT sc_RastPort, rp_SIZEOF ; describing Screen rendering
952: STRUCT sc_BitMap, bm_SIZEOF ; auxiliary graphexcess
baggage
953: STRUCT sc_LayerInfo, li_SIZEOF ; each screen gets a
LayerInfo
954:
955: ; You supply a linked-list of Gadgets for your Screen.
956: ; This list DOES NOT include system Gadgets. You get
the standard
957: ; system Screen Gadgets by default
958: APTR sc_FirstGadget
959:
960: BYTE sc_DetailPen ; for bar/border/gadget rendering
961: BYTE sc_BlockPen ; for bar/border/gadget rendering
962:
963: ; the following variable(s) are maintained by Intuition
to support the
964: ; DisplayBeep() color flashing technique
965: WORD sc_SaveColor0
966:
967: ; This layer is for the Screen and Menu bars
968: APTR sc_BarLayer
969:
970: APTR sc_ExtData
971:
972: APTR sc_UserData ; general-purpose pointer to User
data
973:
974: LABEL sc_SIZEOF
975:
976:
977: ; --- FLAGS SET BY INTUITION -----
978: ; The SCREENTYPE bits are reserved for describing various
Screen types
979: ; available under Intuition.
980: SCREENTYPE equ $000F ; all the screens types available
981: ; --- the definitions for the Screen Type -----
982: WBENCHSCREEN equ $0001 ; Ta Da! The Workbench
983: CUSTOMSCREEN equ $000F ; for that special look
984:
985: SHOWTITLE equ $0010 ; this gets set by a call to ShowTitle()
986:
987: BEEPING equ $0020 ; set when Screen is beeping
988:
989: CUSTOMBITMAP equ $0040 ; if you are supplying your own
BitMap
990:
991:
992:
993:
994: ; =====
995: ; === NewScreen =====
996: ; =====
997: STRUCTURE NewScreen, 0
998:
999: WORD ns_LeftEdge ; initial Screen dimensions
1000: WORD ns_TopEdge ; initial Screen dimensions
1001: WORD ns_Width ; initial Screen dimensions
1002: WORD ns_Height ; initial Screen dimensions
1003: WORD ns_Depth ; initial Screen dimensions
1004:
1005: BYTE ns_DetailPen ; default rendering pens (for
Windows too)
1006: BYTE ns_BlockPen ; default rendering pens (for Windows
too)

```

```

1007:
1008: WORD ns_ViewModes ; display "modes" for this Screen
1009:
1010: WORD ns_Type ; Intuition Screen Type specifier
1011:
1012: APTR ns_Font ; default font for Screen and Windows
1013:
1014: APTR ns_DefaultTitle ; Title when Window doesn't care
1015:
1016: APTR ns_Gadgets ; Your own initial Screen Gadgets
1017:
1018: ; if you are opening a CUSTOMSCREEN and already have
a BitMap
1019: ; that you want used for your Screen, you set the flags
CUSTOMBITMAP in
1020: ; the Types variable and you set this variable to point
to your BitMap
1021: ; structure. The structure will be copied into your
Screen structure,
1022: ; after which you may discard your own BitMap if you
want
1023: APTR ns_CustomBitMap;
1024:
1025: LABEL ns_SIZEOF
1026:
1027:
1028:
1029:
1030:
1031:
1032:
1033:
1034: ; =====
1035: ; == Preferences =====
1036: ; =====
1037:
1038: ; these are the definitions for the printer configurations
1039: FILENAME_SIZE equ 30 ; Filename size
1040:
1041: POINTERSIZE equ (1+16+1)*2 ; Size of Pointer data
buffer
1042:
1043: ; These defines are for the default font size. These actually
describe the
1044: ; height of the defaults fonts. The default font type is
the topaz
1045: ; font, which is a fixed width font that can be used in
either
1046: ; eighty-column or sixty-column mode. The Preferences structure
reflects
1047: ; which is currently selected by the value found in the
variable FontSize,
1048: ; which may have either of the values defined below. These
values actually
1049: ; are used to select the height of the default font. By
changing the
1050: ; height, the resolution of the font changes as well.
1051: TOPAZ_EIGHTY equ 8
1052: TOPAZ_SIXTY equ 9
1053:
1054: ; -----
1055: STRUCTURE Preferences,0
1056:
1057: ; the default font height
1058: BYTE pf_FontHeight ; height for system default font
1059:
1060: ; constant describing what's hooked up to the port
1061: UBYTE pf_PrinterPort ; printer port connection
1062:
1063: ; the baud rate of the port
1064: USHORT pf_BaudRate ; baud rate for the serial port
1065:
1066: ; various timing rates
1067: STRUCT pf_KeyRptSpeed,TV_SIZE ; repeat speed for keyboard
1068: STRUCT pf_KeyRptDelay,TV_SIZE ; Delay before keys repeat
1069: STRUCT pf_DoubleClick,TV_SIZE ; Interval allowed between
clicks
1070:
1071: ; Intuition Pointer data
1072: STRUCT pf_PointerMatrix,POINTERSIZE*2 ; Definition of
pointer sprite
1073: BYTE pf_XOffset ; X-Offset for active 'bit'
1074: BYTE pf_YOffset ; Y-Offset for active 'bit'
1075: WORD pf_color17 ;*****
1076: WORD pf_color18 ; Colours for sprite pointer
1077: WORD pf_color19 ;*****
1078: WORD pf_PointerTicks ; Sensitivity of the pointer
1079:
1080: ; Workbench Screen colors
1081: WORD pf_color0 ;*****
1082: WORD pf_color1 ; Standard default colours
1083: WORD pf_color2 ; Used in the Workbench
1084: WORD pf_color3 ;*****
1085:
1086: ; positioning data for the Intuition View
1087: BYTE pf_ViewXOffset ; Offset for top lefthand corner
1088: BYTE pf_ViewYOffset ; X and Y dimensions
1089: WORD pf_ViewInitX ; View initial offsets at startup
1090: WORD pf_ViewInitY ; View initial offsets at startup
1091:
1092: BOOL EnableCLI ; CLI availability switch
1093:
1094: ; printer configurations
1095: WORD pf_PrinterType ; printer type
1096: STRUCT pf_PrinterFilename,FILENAME_SIZE ; file for printer
1097:
1098: ; print format and quality configurations
1099: SHORT pf_PrintPitch ; print pitch
1100: WORD pf_PrintQuality ; print quality
1101: WORD pf_PrintSpacing ; number of lines per inch
1102: UWORD pf_PrintLeftMargin ; left margin in characters
1103: UWORD pf_PrintRightMargin ; right margin in characters
1104: WORD pf_PrintImage ; positive or negative
1105: WORD pf_PrintAspect ; horizontal or vertical
1106: WORD pf_PrintShade ; b&w, half-tone, or color
1107: WORD pf_PrintThreshold ; darkness ctrl for b/w dumps

```

```

1108:
1109:
1110: ; print paper description
1111: WORD pf_PaperSize ; paper size
1112: UWORD pf_PaperLength ; paper length in lines
1113: WORD pf_PaperType ; continuous or single sheet
1114:
1115: STRUCT pf_padding,50 ; For further system expansion
1116:
1117: LABEL pf_SIZEOF
1118:
1119:
1120: ; === Preferences definitions =====
1121:
1122: ; PrinterPort
1123: PARALLEL_PRINTER equ $00
1124: SERIAL_PRINTER equ $01
1125:
1126: ; BaudRate
1127: BAUD_110 equ $00
1128: BAUD_300 equ $01
1129: BAUD_1200 equ $02
1130: BAUD_2400 equ $03
1131: BAUD_4800 equ $04
1132: BAUD_9600 equ $05
1133: BAUD_19200 equ $06
1134: BAUD_MIDI equ $07
1135:
1136: ; PaperType
1137: FANFOLD equ $00
1138: SINGLE equ $80
1139:
1140: ; PrintPitch
1141: PICA equ $000
1142: ELITE equ $400
1143: FINE equ $800
1144:
1145: ; PrintQuality
1146: DRAFT equ $000
1147: LETTER equ $100
1148:
1149: ; PrintSpacing
1150: SIX_LPI equ $000
1151: EIGHT_LPI equ $200
1152:
1153: ; Print Image
1154: IMAGE_POSITIVE equ 0
1155: IMAGE_NEGATIVE equ 1
1156:
1157: ; PrintAspect
1158: ASPECT_HORIZ equ 0
1159: ASPECT_VERT equ 1
1160:
1161: ; PrintShade
1162: SHADE_BW equ $00
1163: SHADE_GREYSCALE equ $01
1164: SHADE_COLOR equ $02
1165:
1166: ; PaperSize
1167: US_LETTER equ $00

```

```

1168: US_LEGAL equ $10
1169: N_TRACTOR equ $20
1170: W_TRACTOR equ $30
1171: CUSTOM equ $40
1172:
1173: ; PrinterType
1174: CUSTOM_NAME equ $00
1175: ALPHA_P_101 equ $01
1176: BROTHER_15XL equ $02
1177: CBM_MPS1000 equ $03
1178: DIAB_630 equ $04
1179: DIAB_ADV_D25 equ $05
1180: DIAB_C_150 equ $06
1181: EPSON equ $07
1182: EPSON_JX_80 equ $08
1183: OKIMATE_20 equ $09
1184: QUME_LP_20 equ $0A
1185: ; new printer entries, 3 October 1985
1186: HP_LASERJET equ $0B
1187: HP_LASERJET_PLUS equ $0C
1188:
1189:
1190:
1191:
1192: ; =====
1193: ; === Remember =====
1194: ;
1195: ; this structure is used for remembering what memory has
; been allocated to
1196: ; date by a given routine, so that a premature abort or
; systematic exit
1197: ; can deallocate memory cleanly, easily, and completely
1198: STRUCTURE Remember,0
1199:
1200: APTR rm_NextRemember
1201: ULONG rm_RememberSize
1202: APTR rm_Memory
1203:
1204: LABEL rm_SIZEOF
1205:
1206:
1207:
1208: ; =====
1209: ; === Miscellaneous =====
1210: ; =====
1211:
1212: ; = MACROS =====
1213: ;#define MENUNUM(n) (n & 0x1F)
1214: ;#define ITEMNUM(n) ((n >> 5) & 0x003F)
1215: ;#define SUBNUM(n) ((n >> 11) & 0x001F)
1216: ;
1217: ;#define SHIFTMENU(n) (n & 0x1F)
1218: ;#define SHIFTTITEM(n) ((n & 0x3F) << 5)
1219: ;#define SHIFTSUB(n) ((n & 0x1F) << 11)
1220: ;
1221: ; = MENU STUFF =====

```



```

1222: NOMENU equ $001F
1223: NOITEM equ $003F
1224: NOSUB equ $001F
1225: MENUNULL equ $FFFF
1226:
1227:
1228: ; = -RJ='s peculiarities =====

1229: ;#define FOREVER for(;;)
1230: ;#define SIGN(x) (((x) > 0) - ((x) < 0))
1231:
1232:
1233: ; these defines are for the COMMSEQ and CHECKIT menu stuff.
 If CHECKIT,
1234: ; I'll use a generic Width (for all resolutions) for the
 CheckMark.
1235: ; If COMMSEQ, likewise I'll use this generic stuff
1236: CHECKWIDTH equ 19
1237: COMMWIDTH equ 27
1238: LOWCHECKWIDTH equ 13
1239: LOWCOMMWIDTH equ 16
1240:
1241:
1242: ; these are the AlertNumber defines. if you are calling
 DisplayAlert()
1243: ; the AlertNumber you supply must have the ALERT_TYPE bits
 set to one
1244: ; of these patterns
1245: ALERT_TYPE equ $80000000
1246: RECOVERY_ALERT equ $00000000 ; the system can recover
 from this
1247: DEADEND_ALERT equ $80000000 ; no recovery possible,
 this is it

1248:
1249:
1250: ; When you're defining IntuiText for the Positive and Negative
 Gadgets
1251: ; created by a call to AutoRequest(), these defines will
 get you
1252: ; reasonable-looking text. The only field without a define
 is the IText
1253: ; field; you decide what text goes with the Gadget
1254: AUTOFRONTPEN equ 0
1255: AUTOBACKPEN equ 1
1256: AUTODRAWMODE equ RP_JAM2
1257: AUTOLEFTEDGE equ 6
1258: AUTOTOPEGE equ 3
1259: AUTOTEXTFONT equ 0
1260: AUTONEXTTEXT equ 0
1261:
1262:
1263:
1264: ENDC

```

```

1: IFND INTUITION_INTUITIONBASE_I
2: INTUITION_INTUITIONBASE_I SET 1
3:
4: *** intuitionbase.i *****/
5: * Commodore-Amiga, Inc.
6: *
7: * the IntuitionBase structure and supporting structures
8: *
9: * Modification History
10: * date : author : Comments
11: *
12: * 3-1-85 -jimm
13: *
14: *****/
15:
16: IFND EXEC_LIBRARIES_I
17: INCLUDE "exec/libraries.i"
18: ENDC
19:
20: IFND GRAPHICS_VIEW_I
21: INCLUDE "graphics/view.i"
22: ENDC
23:
24: * Be sure to protect yourself against someone modifying
 these data as
25: * you look at them. This is done by calling:
26: *
27: * lock = LockIBase(0), which returns a ULONG. When done
 call
28: * D0 D0
29: * UnlockIBase(lock) where lock is what LockIBase() returned.
30: * A0
31: * NOTE: these library functions are simply stubs now, but
 should be called
32: * to be compatible with future releases.
33:
34: * =====
35: * IntuitionBase =====
36: *
37: STRUCTURE IntuitionBase,0
38:
39: STRUCT ib_LibNode,LIB_SIZE
40: STRUCT ib_ViewLord,SIZEOF_VIEW
41: APTR ib_ActiveWindow
42: APTR ib_ActiveScreen
43:
44: * the FirstScreen variable points to the frontmost Screen.
 Screens are
45: * then maintained in a front to back order using Screen.NextScreen
46:
47: APTR ib_FirstScreen
48:
49: * there is not size here because...
50: *
51: *
52: ENDC

```

Contents

libraries/diskfont.i  
 libraries/dos.i  
 libraries/dos\_lib.i  
 libraries/dosextens.i  
 libraries/translator.i

```

1 IFND LIBRARIES_DISKFONT_I
2 LIBRARIES_DISKFONT_I SET 1
3 *****
4 * Commodore-Amiga, Inc. *
5 * diskfont.i *
6 *****
7 *****
8 *
9 * diskfont library definitions
10 *
11 *****
12
13 IFND EXEC_NODES_I
14 INCLUDE "exec/nodes.i"
15 ENDC
16 IFND EXEC_LISTS_I
17 INCLUDE "exec/lists.i"
18 ENDC
19 IFND GRAPHICS_TEXT_I
20 INCLUDE "graphics/text.i"
21 ENDC
22
23 MAXFONTPATH EQU 256 ; including null terminator
24
25 STRUCTURE FC,0
26 STRUCT fc_FileName,MAXFONTPATH
27 UWORD fc_ysize
28 UBYTE fc_Style
29 UBYTE fc_Flags
30 LABEL fc_SIZEOF
31
32 FCH_ID EQU $0f00
33
34 STRUCTURE FCH,0
35 UWORD fch_FileID ; FCH_ID
36 UWORD fch_NumEntries ; the number of FontContents elements
37 LABEL fch_FC ; the FontContents elements
38
39
40 DFH_ID EQU $0f80
41 MAXFONTNAME EQU 32 ; font name including ".font\0"
42
43 STRUCTURE DiskFontHeader,0
44 ; the following 8 bytes are not actually considered a part of the
45 ; DiskFontHeader, but immediately precede it. The NextSegment is supplied
46 ; by the linker/loader, and the ReturnCode is the code at the beginning
47 ; of the font in case someone runs it...
48 ; ULONG dfh_NextSegment ; actually a BPTR
49 ; ULONG dfh_ReturnCode ; MOVEQ #0,D0 : RTS
50 ; here then is the official start of the DiskFontHeader...
51 STRUCT dfh_DF,LN_SIZE ; node to link disk fonts
52 UWORD dfh_FileID ; DFH_ID
53 UWORD dfh_Revision ; the font revision in this version
54 LONG dfh_Segment ; the segment address when loaded
55 STRUCT dfh_Name,MAXFONTNAME ; the font name (null terminated)
56 STRUCT dfh_TF,tf_SIZEOF ; loaded TextFont structure
57 LABEL dfh_SIZEOF
58
59

```

```

60 BITDEF AF, MEMORY, 0
61 BITDEF AF, DISK, 1
62
63 STRUCTURE AF, 0
64 UWORD af_Type ; MEMORY or DISK
65 STRUCT af_Attr, ta_SIZEOF ; text attributes for font
66 LABEL af_SIZEOF
67
68 STRUCTURE AFH, 0
69 UWORD afh_NumEntries ; number of AvailFonts elements
70 LABEL afh_AF ; the AvailFonts elements
71
72 ENDC

```

```

1 IFND LIBRARIES_DOS_I
2 LIBRARIES_DOS_I SET 1
3
4 *****
5 * Commodore-Amiga, Inc. *
6 * dos.i *
7 *****
8 * Standard assembler header for Amiga DOS on the MC68000
9
10 * IFND EXEC_TYPES_I
11 * INCLUDE "exec/types.i"
12 * ENDC
13
14
15 DOSNAME MACRO
16 DC.B 'dos.library', 0
17 ENDM
18
19 * Predefined Amiga DOS global constants
20
21 * Mode parameter to Open()
22 MODE_OLDFILE EQU 1005 * Open existing file read/write
23 * * positioned at beginning of file.
24 MODE_NEWFILE EQU 1006 * Open freshly created file (delete
25 * * old file) read/write
26
27 * Relative position to Seek()
28 OFFSET_BEGINNING EQU -1 * relative to Beginning Of File
29 OFFSET_CURRENT EQU 0 * relative to Current file position
30 OFFSET_END EQU 1 * relative to End Of File
31
32 OFFSET_BEGINNING EQU OFFSET_BEGINNING * Ancient compatibility
33
34 BITSPERBYTE EQU 8
35 BYTESPERLONG EQU 4
36 BITSPERLONG EQU 32
37 MAXINT EQU $7FFFFFFF
38 MININT EQU $80000000
39
40 * Passed as type to Lock()
41 SHARED_LOCK EQU -2 ; File is readable by others
42 ACCESS_READ EQU -2 ; Synonym
43 EXCLUSIVE_LOCK EQU -1 ; No other access allowed
44 ACCESS_WRITE EQU -1 ; Synonym
45
46
47 STRUCTURE DateStamp, 0
48 LONG ds_Days ; Number of days since Jan. 1, 1978
49 LONG ds_Minute ; Number of minutes past midnight
50 LONG ds_Tick ; Number of ticks past minute
51 LABEL ds_SIZEOF ; DateStamp
52 TICKS_PER_SECOND EQU 50 ; Number of ticks in one second
53
54 * Returned by Examine() and ExInfo()
55 STRUCTURE FileInfoBlock, 0
56 LONG fib_DiskKey
57 LONG fib_DirEntryType ; Type of Directory. If <0, then a plain fil
58 ; If > 0 then a directory
59 STRUCT fib_FileName, 108 ; Null terminated. Max 30 chars used for now

```

```

60 LONG fib_Protection ; bit mask of protection, rwx are 3-0.
61 LONG fib_EntryType
62 LONG fib_Size ; Number of bytes in file
63 LONG fib_NumBlocks ; Number of blocks in file
64 STRUCT fib_DateStamp,ds_SIZEOF ; Date file last changed.
65 STRUCT fib_Comment,116 ; Null terminated.
66 ; Comment associated with file
67 LABEL fib_SIZEOF ; FileInfoBlock
68
69 * FIB stands for FileInfoBlock
70 * FIBB are bit definitions, FIBF are field definitions
71 BITDEF FIB_READ,3
72 BITDEF FIB_WRITE,2
73 BITDEF FIB_EXECUTE,1
74 BITDEF FIB_DELETE,0
75
76
77 * All BCPL data must be long word aligned. BCPL pointers are the long word
78 * address (i.e byte address divided by 4 (>>2))
79
80 * Macro to indicate BCPL pointers
81 BPTR MACRO ; Long word pointer
82 LONG \1
83 ENDM
84 BSTR MACRO ; Long word pointer to BCPL string.
85 LONG \1
86 ENDM
87
88 * #define BADDR(bptr) (bptr << 2) * Convert BPTR to byte addressed pointer
89
90 * BCPL strings have a length in the first byte and then the characters.
91 * For example: s[0]=3 s[1]=S s[2]=Y s[3]=S
92
93 * returned by Info()
94 STRUCTURE InfoData,0
95 LONG id_NumSoftErrors * number of soft errors on disk
96 LONG id_UnitNumber * Which unit disk is (was) mounted on
97 LONG id_DiskState * See defines below
98 LONG id_NumBlocks * Number of blocks on disk
99 LONG id_NumBlocksUsed * Number of block in use
100 LONG id_BytesPerBlock
101 LONG id_DiskType * Disk Type code
102 BPTR id_VolumeNode * BCPL pointer to volume node
103 LONG id_InUse * Flag, zero if not in use
104 LABEL id_SIZEOF * InfoData
105
106 * ID stands for InfoData
107 * Disk states
108 ID_WRITE_PROTECTED EQU 80 * Disk is write protected
109 ID_VALIDATING EQU 81 * Disk is currently being validated
110 ID_VALIDATED EQU 82 * Disk is consistent and writeable
111 * Disk types
112 ID_NO_DISK_PRESENT EQU -1
113 ID_UNREADABLE_DISK EQU ('B'<<24)!('A'<<16)!('D'<<8)
114 ID_NOT_REALLY_DOS EQU ('N'<<24)!('D'<<16)!('O'<<8)!('S')
115 ID_DOS_DISK EQU ('D'<<24)!('O'<<16)!('S'<<8)
116 ID_KICKSTART_DISK EQU ('K'<<24)!('I'<<16)!('C'<<8)!('K')
117
118 * Errors from IoErr(), etc.
119 ERROR_NO_FREE_STORE EQU 103

```

```

120 ERROR_OBJECT_IN_USE EQU 202
121 ERROR_OBJECT_EXISTS EQU 203
122 ERROR_OBJECT_NOT_FOUND EQU 205
123 ERROR_ACTION_NOT_KNOWN EQU 209
124 ERROR_INVALID_COMPONENT_NAME EQU 210
125 ERROR_INVALID_LOCK EQU 211
126 ERROR_OBJECT_WRONG_TYPE EQU 212
127 ERROR_DISK_NOT_VALIDATED EQU 213
128 ERROR_DISK_WRITE_PROTECTED EQU 214
129 ERROR_RENAME_ACROSS_DEVICES EQU 215
130 ERROR_DIRECTORY_NOT_EMPTY EQU 216
131 ERROR_DEVICE_NOT_MOUNTED EQU 218
132 ERROR_SEEK_ERROR EQU 219
133 ERROR_COMMENT_TOO_BIG EQU 220
134 ERROR_DISK_FULL EQU 221
135 ERROR_DELETE_PROTECTED EQU 222
136 ERROR_WRITE_PROTECTED EQU 223
137 ERROR_READ_PROTECTED EQU 224
138 ERROR_NOT_A_DOS_DISK EQU 225
139 ERROR_NO_DISK EQU 226
140 ERROR_NO_MORE_ENTRIES EQU 232
141
142 * These are the return codes used by convention by AmigaDOS commands
143 * See FAILAT and IF for relevance to EXECUTE files
144 RETURN_OK EQU 0 * No problems, success
145 RETURN_WARN EQU 5 * A warning only
146 RETURN_ERROR EQU 10 * Something wrong
147 RETURN_FAIL EQU 20 * Complete or severe failure
148
149 * Bit numbers that signal you that a user has issued a break
150 BITDEF SIGBREAK,CTRL_C,12
151 BITDEF SIGBREAK,CTRL_D,13
152 BITDEF SIGBREAK,CTRL_E,14
153 BITDEF SIGBREAK,CTRL_F,15
154
155

```

```

1 *****
2 * Commodore-Amiga, Inc. *
3 * dos_lib.i *
4 *****
5 *
6 * Library interface offsets for DOS library
7 *
8 reserve EQU 4
9 vsize EQU 6
10 count SET -vsize*(reserve+1)
11 LIBENT MACRO
12 _LVO\1 EQU count
13 count SET count-vsize
14 ENDM
15 *
16 *
17 *
18 LIBENT Open
19 LIBENT Close
20 LIBENT Read
21 LIBENT Write
22 LIBENT Input
23 LIBENT Output
24 LIBENT Seek
25 LIBENT DeleteFile
26 LIBENT Rename
27 LIBENT Lock
28 LIBENT UnLock
29 LIBENT DupLock
30 LIBENT Examine
31 LIBENT ExNext
32 LIBENT Info
33 LIBENT CreateDir
34 LIBENT CurrentDir
35 LIBENT IoErr
36 LIBENT CreateProc
37 LIBENT Exit
38 LIBENT LoadSeg
39 LIBENT UnLoadSeg
40 LIBENT GetPacket
41 LIBENT QueuePacket
42 LIBENT DeviceProc
43 LIBENT SetComment
44 LIBENT SetProtection
45 LIBENT DateStamp
46 LIBENT Delay
47 LIBENT WaitForChar
48 LIBENT ParentDir
49 LIBENT IsInteractive
50 LIBENT Execute

```

```

1 IFND LIBRARIES_DOSEXTENS_I
2
3 LIBRARIES_DOSEXTENS_I SET 1
4 *****
5 * Commodore-Amiga, Inc. *
6 * dosextens.i *
7 *****
8 * DOS structures not needed for the casual DOS user
9
10
11 IFND EXEC_TYPES_I
12 INCLUDE "exec/types.i"
13 ENDC
14 IFND EXEC_TASKS_I
15 INCLUDE "exec/tasks.i"
16 ENDC
17 IFND EXEC_PORTS_I
18 INCLUDE "exec/ports.i"
19 ENDC
20 IFND EXEC_LIBRARIES_I
21 INCLUDE "exec/libraries.i"
22 ENDC
23
24 IFND LIBRARIES_DOS_I
25 INCLUDE "libraries/dos.i"
26 ENDC
27
28
29 * All DOS processes have this STRUCTURE
30 * Create and DeviceProc returns pointer to the MsgPort in this STRUCTURE
31 * Process_addr = DeviceProc(..) - TC_SIZE
32
33 STRUCTURE Process,0
34 STRUCT pr_Task,TC_SIZE
35 STRUCT pr_MsgPort,MP_SIZE * This is BPTR address from DOS functions
36 WORD pr_Pad * Remaining variables on 4 byte boundaries
37 BPTR pr_SegList * Array of seg lists used by this process
38 LONG pr_StackSize * Size of process stack in bytes
39 APTR pr_GlobVec * Global vector for this process (BCPL)
40 LONG pr_TaskNum * CLI task number of zero if not a CLI
41 BPTR pr_StackBase * Ptr to high memory end of process stack
42 LONG pr_Result2 * Value of secondary result from last call
43 BPTR pr_CurrentDir * Lock associated with current directory
44 BPTR pr_CIS * Current CLI Input Stream
45 BPTR pr_COS * Current CLI Output Stream
46 APTR pr_ConsoleTask * Console handler process for current window
47 APTR pr_FileSystemTask * File handler process for current drive
48 BPTR pr_CLI * pointer to ConsoleLineInterpreter
49 APTR pr_ReturnAddr * pointer to previous stack frame
50 APTR pr_PktWait * Function to be called when awaiting msg
51 APTR pr_WindowPtr * Window pointer for errors
52 LABEL pr_SIZEOF * Process
53
54 * The long word address (BPTR) of this STRUCTURE is returned by
55 * Open() and other routines that return a file. You need only worry
56 * about this STRUCT to do async io's via PutMsg() instead of
57 * standard file system calls
58
59 STRUCTURE FileHandle,0

```

```

60 APTR fh_Link * pointer to EXEC message
61 APTR fh_Interactive * Boolean; TRUE if interactive handle
62 APTR fh_Type * Port to do PutMsg() to
63 LONG fh_Buf
64 LONG fh_Pos
65 LONG fh_End
66 LONG fh_Funcs
67 fh_Func1 EQU fh_Funcs
68 LONG fh_Func2
69 LONG fh_Func3
70 LONG fh_Args
71 fh_Arg1 EQU fh_Args
72 LONG fh_Arg2
73 LABEL fh_SIZEOF * FileHandle
74
75 * This is the extension to EXEC Messages used by DOS
76 STRUCTURE DosPacket,0
77 APTR dp_Link * pointer to EXEC message
78 APTR dp_Port * pointer to Reply port for the packet
79 * * Must be filled in each send.
80 LONG dp_Type * See ACTION_... below and
81 * * 'R' means Read, 'W' means Write to the file system
82 LONG dp_Res1 * For file system calls this is the result
83 * * that would have been returned by the
84 * * function, e.g. Write ('W') returns actual
85 * * length written
86 LONG dp_Res2 * For file system calls this is what would
87 * * have been returned by IoErr()
88 LONG dp_Arg1
89 * Device packets common equivalents
90 dp_Action EQU dp_Type
91 dp_Status EQU dp_Res1
92 dp_Status2 EQU dp_Res2
93 dp_BufAddr EQU dp_Arg1
94 LONG dp_Arg2
95 LONG dp_Arg3
96 LONG dp_Arg4
97 LONG dp_Arg5
98 LONG dp_Arg6
99 LONG dp_Arg7
100 LABEL dp_SIZEOF * DosPacket
101
102 * A Packet does not require the Message to before it in memory, but
103 * for convenience it is useful to associate the two.
104 * Also see the function init_std_pkt for initializing this STRUCTURE
105
106 STRUCTURE StandardPacket,0
107 STRUCT sp_Msg,MN_SIZE
108 STRUCT sp_Pkt,dp_SIZEOF
109 LABEL sp_SIZEOF * StandardPacket
110
111
112 * Packet types
113 ACTION_NIL EQU 0
114 ACTION_GET_BLOCK EQU 2
115 ACTION_SET_MAP EQU 4
116 ACTION_DIE EQU 5
117 ACTION_EVENT EQU 6
118 ACTION_CURRENT_VOLUME EQU 7
119 ACTION_LOCATE_OBJECT EQU 8

120 ACTION_RENAME_DISK EQU 9
121 ACTION_WRITE EQU 'W'
122 ACTION_READ EQU 'R'
123 ACTION_FREE_LOCK EQU 15
124 ACTION_DELETE_OBJECT EQU 16
125 ACTION_RENAME_OBJECT EQU 17
126
127 ACTION_COPY_DIR EQU 19
128 ACTION_WAIT_CHAR EQU 20
129 ACTION_SET_PROTECT EQU 21
130 ACTION_CREATE_DIR EQU 22
131 ACTION_EXAMINE_OBJECT EQU 23
132 ACTION_EXAMINE_NEXT EQU 24
133 ACTION_DISK_INFO EQU 25
134 ACTION_INFO EQU 26
135
136 ACTION_SET_COMMENT EQU 28
137 ACTION_PARENT EQU 29
138 ACTION_TIMER EQU 30
139 ACTION_INHIBIT EQU 31
140 ACTION_DISK_TYPE EQU 32
141 ACTION_DISK_CHANGE EQU 33
142
143
144 * DOS library node structure.
145 * This is the data at positive offsets from the library node.
146 * Negative offsets from the node is the jump table to DOS functions
147 * node = (STRUCTURE DosLibrary *) OpenLibrary("dos.library" ..)
148
149 STRUCTURE DosLibrary,0
150 STRUCT dl_lib,LIB_SIZE
151 APTR dl_Root * Pointer to RootNode, described below
152 APTR dl_GV * Pointer to BCPL global vector
153 LONG dl_A2 * Private register dump of DOS
154 LONG dl_A5
155 LONG dl_A6
156 LABEL dl_SIZEOF * DosLibrary
157
158 *
159
160 STRUCTURE RootNode,0
161 BPTR rn_TaskArray * [0] is max number of CLI's
162 * * [1] is APTR to process id of CLI 1
163 * * [n] is APTR to process id of CLI n
164 BPTR rn_ConsoleSegment * SegList for the CLI
165 STRUCT rn_Time,ds_SIZEOF * Current time
166 LONG rn_RestartSeg * SegList for the disk validator process
167 BPTR rn_Info * Pointer ot the Info structure
168 LABEL rn_SIZEOF * RootNode
169
170 STRUCTURE DosInfo,0
171 BPTR di_McName * Network name of this machine currently 0
172 BPTR di_DevInfo * Device List
173 BPTR di_Devices * Currently zero
174 BPTR di_Handlers * Currently zero
175 APTR di_NetHand * Network handler processid currently zero
176 LABEL di_SIZEOF * DosInfo
177
178 * DOS Processes started from the CLI via RUN or NEWCLI have this additional
179 * set to data associated with them

```

```

180
181 STRUCTURE CommandLineInterface,0
182 LONG cli_Result2 * Value of IoErr from last command
183 BSTR cli_SetName * Name of current directory
184 BPTR cli_CommandDir * Lock associated with command directory
185 LONG cli_ReturnCode * Return code from last command
186 BSTR cli_CommandName * Name of current command
187 LONG cli_FailLevel * Fail level (set by FAILAT)
188 BSTR cli_Prompt * Current prompt (set by PROMPT)
189 BPTR cli_StandardInput * Default (terminal) CLI input
190 BPTR cli_CurrentInput * Current CLI input
191 BSTR cli_CommandFile * Name of EXECUTE command file
192 LONG cli_Interactive * Boolean True if prompts required
193 LONG cli_Background * Boolean True if CLI created by RUN
194 BPTR cli_CurrentOutput * Current CLI output
195 LONG cli_DefaultStack * Stack size to be obtained in long words
196 BPTR cli_StandardOutput * Default (terminal) CLI output
197 BPTR cli_Module * SegList of currently loaded command
198 LABEL cli_SIZEOF * CommandLineInterface
199
200
201 *

```

```

202 * this structure needs some work. It should really be a union, because
203 * it can take on different valued depending on whether it is a device,
204 * an assigned directory, or a volume.
205 * For now, it reflects a volume.
206 *

```

```

207 STRUCTURE DevList,0
208 BPTR dl_Next ; bptr to next device list
209 LONG dl_Type ; see DLT below
210 APTR dl_Task ; ptr to handler task
211 BPTR dl_Lock ; not for volumes
212 STRUCT dl_VolumeDate,ds_SIZEOF ; creation date
213 BPTR dl_LockList ; outstanding locks
214 LONG dl_DiskType ; 'DOS', etc
215 LONG dl_unused
216 BSTR dl_Name ; bptr to bcpl name
217 LABEL DevList_SIZEOF
218
219 * definitions for dl_Type
220 DLT_DEVICE EQU 0
221 DLT_DIRECTORY EQU 1
222 DLT_VOLUME EQU 2
223
224
225 * a lock structure, as returned by Lock() or DupLock()
226 STRUCTURE FileLock,0
227 BPTR fl_Link ; bcpl pointer to next lock
228 LONG fl_Key ; disk block number
229 LONG fl_Access ; exclusive or shared
230 APTR fl_Task ; handler task's port
231 BPTR fl_Volume ; bptr to a DeviceList
232 LABEL fl_SIZEOF
233
234

```

```

1 IFND LIBRARIES_TRANSLATOR_I
2 LIBRARIES_TRANSLATOR_I SET 1
3 *****
4 * Commodore-Amiga, Inc. *
5 * translator.i *
6 *****
7
8
9 * Translator error codes
10 TR_NotUsed EQU -1 ;This is an often used system rc
11 TR_NoMem EQU -2 ;Can't allocate memory
12 TR_MakeBad EQU -4 ;Error in MakeLibrary call
13
14
15 ENDC

```

D - 78

Contents

resources/ciabase.i  
resources/disk.i  
resources/misc.i  
resources/potgo.i

```
1: *****
2: * Commodore-Amiga, Inc.
 *
3: * ciabase.i
 *
4: *****
5:
6: *-----
7: *
8: * CIA Resource Data Definition
9: *
10: *-----
11:
12:
13: STRUCTURE CIAR,LIB_SIZE
14: APTR CR_HWADDR
15: UWORD CR_IntMask
16: UBYTE CR_IEnable
17: UBYTE CR_IActive
18: STRUCT CR_INTNODE,IS_SIZE
19: STRUCT CR_IVTA,IV_SIZE
20: STRUCT CR_IVTB,IV_SIZE
21: STRUCT CR_IVALRM,IV_SIZE
22: STRUCT CR_IVSP,IV_SIZE
23: STRUCT CR_IVFLG,IV_SIZE
24: LABEL CR_SIZE
25:
```



```

1 IFND RESOURCES_DISK_I
2 RESOURCES_DISK_I SET 1
3 *****
4 * Commodore-Amiga, Inc. *
5 * disk.i *
6 *****
7 *****
8 *****
10 * external declarations for disk resources
11 *
12 * SOURCE CONTROL
13 * -----
14 * $Header: disk.i,v 27.3 85/07/12 23:17:43 neil Exp $
15 * $Locker: $
16 *
17 *****
18 *****
19 *****
20 IFND EXEC_TYPES_I
21 INCLUDE "exec/types.i"
22 *****
23 *****
24 IFND EXEC_LISTS_I
25 INCLUDE "exec/lists.i"
26 *****
27 *****
28 IFND EXEC_PORTS_I
29 INCLUDE "exec/ports.i"
30 *****
31 *****
32 IFND EXEC_INTERRUPTS_I
33 INCLUDE "exec/interrupts.i"
34 *****
35 *****
36 IFND EXEC_LIBRARIES_I
37 INCLUDE "exec/libraries.i"
38 *****
39 *****
40 *****
41 *****
42 *****
43 * Resource structures
44 *****
45 *****
46 *****
47 STRUCTURE DISCRESOURCEUNIT,MN_SIZE
48 STRUCT DRU_DISCBLOCK,IS_SIZE
49 STRUCT DRU_DISCSYNC,IS_SIZE
50 STRUCT DRU_INDEX,IS_SIZE
51 LABEL DRU_SIZE
52 *****
53 *****
54 *****
55 STRUCTURE DISCRESOURCE,LIB_SIZE
56 APTR DR_CURRENT ; pointer to current unit structure
57 UBYTE DR_FLAGS
58 UBYTE DR_pad
59 APTR DR_SYSLIB
60 APTR DR_CIARESOURCE
61 STRUCT DR_UNITID,4*4
62 STRUCT DR_WAITING,LH_SIZE
63 STRUCT DR_DISCBLOCK,IS_SIZE
64 STRUCT DR_DISCSYNC,IS_SIZE
65 STRUCT DR_INDEX,IS_SIZE
66 LABEL DR_SIZE
67 *****
68 BITDEF DR_ALLOC0,0 ; unit zero is allocated
69 BITDEF DR_ALLOC1,1 ; unit one is allocated
70 BITDEF DR_ALLOC2,2 ; unit two is allocated
71 BITDEF DR_ALLOC3,3 ; unit three is allocated
72 BITDEF DR_ACTIVE,7 ; is the disk currently busy?
73 *****
74 *****
75 *****
76 *****
77 * Hardware Magic
78 *****
79 *****
80 *****
81 *****
82 DSKDMAOFF EQU $4000 ; idle command for dsklen register
83 *****
84 *****
85 *****
86 *****
87 * Resource specific commands
88 *****
89 *****
90 *****
91 *-- DR_NAME is a generic macro to get the name of the resource. This
92 *-- way if the name is ever changed you will pick up the change
93 *-- automatically.
94 *--
95 *-- Normal usage would be:
96 *--
97 *-- internalName: DISKNAME
98 *--
99 *****
100 DISKNAME: MACRO
101 DC.B 'disk.resource',0
102 DS.W 0
103 ENDM
104 *****
105 LIBINIT LIB_BASE
106 LIBDEF DR_ALLOCUNIT
107 LIBDEF DR_FREEUNIT
108 LIBDEF DR_GETUNIT
109 LIBDEF DR_GIVEUNIT
110 LIBDEF DR_GETUNITID
111 *****
112 DR_LASTCOMM EQU DR_GIVEUNIT
113 *****
114 *****
115 *****
116 *****
117 * drive types
118 *****
119 *****
120 *****
121 DRT_AMIGA EQU $00000000
122 DRT_37422D2S EQU $55555555
123 DRT_EMPTY EQU $FFFFFFF

```

```

1 IFND RESOURCES_MISC_I
2 RESOURCES_MISC_I SET 1
3
4
5 IFND EXEC_TYPES_I
6 INCLUDE "exec/types.i"
7
8
9 IFND EXEC_LIBRARIES_I
10 INCLUDE "exec/libraries.i"
11
12
13 *****
14 * Commodore-Amiga, Inc. *
15 * misc.i *
16 *****
17
18 *****
19 *
20 * external declarations for misc system resources
21 *
22 * SOURCE CONTROL
23 * -----
24 * $Header: misc.i,v 27.3 85/07/12 16:29:36 neil Exp $
25 *
26 * $Locker: $
27 *
28 *****
29
30
31 *****
32 *
33 * Resource structures
34 *
35 *****
36
37 MR_SERIALPORT EQU 0
38 MR_SERIALBITS EQU 1
39 MR_PARALLELPORT EQU 2
40 MR_PARALLELBITS EQU 3
41
42 NUMMRYPES EQU 4
43
44 STRUCTURE MiscResource,LIB_SIZE
45 STRUCT mr_AllocArray,4*NUMMRYPES
46 LABEL mr_Sizeof
47
48 LIBINIT LIB_BASE
49 LIBDEF MR_ALLOCMISCRESOURCE
50 LIBDEF MR_FREEMISCRESOURCE
51
52
53 MISCNAME MACRO
54 DC.B 'misc.resource',0
55 ENDM
56
57

```

```

1 IFND RESOURCES_POTGO_I
2 RESOURCES_POTGO_I EQU 1
3 *****
4 * Commodore-Amiga, Inc. *
5 * potgo.i *
6 *****
7 POTGONAME MACRO
8 DC.B 'potgo.resource'
9 DC.B 0
10 DS.W 0
11 ENDM
12 ENDC

```

Contents

workbench/icon.i  
workbench/startup.i  
workbench/workbench.i

```
1
2 IFND WORKBENCH_ICON_I
3 WORKBENCH_ICON_I SET 1
4
5 *****
6 * Commodore-Amiga, Inc. *
7 * icon.i *
8 *****
9
10 *****
11 *
12 * icon.i -- external declarations for workbench support library
13 *
14 * SOURCE CONTROL
15 * -----
16 * $Header: icon.i,v 29.1 85/08/07 22:27:14 neil Exp $
17 *
18 * $Locker: $
19 *
20 *****
21
22
23 *****
24 *
25 * Library structures
26 *
27 *****
28
29
30 ICONNAME MACRO
31 DC.B 'icon.library',0
32 ENDM
33
34
```

```

1
2 *** startup.i *****
3 *
4 * Workbench startup definitions
5 *
6 * Commodore-Amiga, Inc.
7 *
8 * $Header: startup.i,v 29.1 85/08/15 06:58:52 neil Exp $
9 *
10 * $Locker: $
11 *
12 *****
13
14 IFND EXEC_TYPES_I
15 INCLUDE "exec/types.i"
16
17
18 IFND EXEC_PORTS_I
19 INCLUDE "exec/ports.i"
20
21
22 IFND LIBRARIES_DOS_I
23 INCLUDE "libraries/dos.i"
24
25
26 STRUCTURE WBStartup,0
27 STRUCT sm_Message,MN_SIZE ; a standard message structure
28 APTR sm_Process ; the process descriptor for you
29 BPTR sm_Segment ; a descriptor for your code
30 LONG sm_NumArgs ; the number of elements in ArgList
31 APTR sm_ToolWindow ; description of window
32 APTR sm_ArgList ; the arguments themselves
33 LABEL sm_SIZEOF
34
35 STRUCTURE WBArg,0
36 BPTR wa_Lock ; a lock descriptor
37 APTR wa_Name ; a string relative to that lock
38 LABEL wa_SIZEOF
39

```

```

1 *****
2 *
3 *
4 * workbench.h
5 *
6 * Commodore-Amiga, Inc.
7 *
8 * $Header: workbench.i,v 31.2 85/09/02 21:32:18 neil Exp $
9 *
10 * $Locker: $
11 *
12 *****
13
14 IFND EXEC_TYPES_I
15 INCLUDE "exec/types.i"
16
17
18 IFND EXEC_NODES_I
19 INCLUDE "exec/nodes.i"
20
21
22 IFND EXEC_LISTS_I
23 INCLUDE "exec/lists.i"
24
25
26 IFND EXEC_TASKS_I
27 INCLUDE "exec/tasks.i"
28
29
30 IFND INTUITION_INTUITION_I
31 INCLUDE "intuition/intuition.i"
32
33
34
35 ; the Workbench object types
36 WBDISK EQU 1
37 WBDRAWER EQU 2
38 WBTOOL EQU 3
39 WBPROJECT EQU 4
40 WBGARBAGE EQU 5
41 WBDEVICE EQU 6
42 WBKICK EQU 7
43
44
45 ; the main workbench object structure
46 STRUCTURE DrawerData,0
47 STRUCT dd_NewWindow,nw_SIZE ; args to open window
48 LONG dd_CurrentX ; current x coordinate of origin
49 LONG dd_CurrentY ; current y coordinate of origin
50 LONG dd_MinX ; smallest x coordinate in window
51 LONG dd_MinY ; smallest y coordinate in window
52 LONG dd_MaxX ; largest x coordinate in window
53 LONG dd_MaxY ; largest y coordinate in window
54 STRUCT dd_HorizScroll,gg_SIZEOF
55 STRUCT dd_VertScroll,gg_SIZEOF
56 STRUCT dd_UpMove,gg_SIZEOF
57 STRUCT dd_DownMove,gg_SIZEOF
58 STRUCT dd_LeftMove,gg_SIZEOF
59 STRUCT dd_RightMove,gg_SIZEOF

```

```

60 STRUCT dd_HorizImage,ig_SIZEOF
61 STRUCT dd_VertImage,ig_SIZEOF
62 STRUCT dd_HorizProp,pi_SIZEOF
63 STRUCT dd_VertProp,pi_SIZEOF
64 APTR dd_DrawerWin ; pointer to drawers window
65 APTR dd_Object ; back pointer to drawer object
66 STRUCT dd_Children,LN_SIZE ; where our children hang out
67 LONG dd_Lock
68 LABEL dd_SIZEOF
69
70 ; the amount of DrawerData actually written to disk
71 DRAWERDATAFILESIZE EQU (nw_SIZE+2*(4))
72
73
74 STRUCTURE DiskObject,0
75 UWORD do_Magic ; a magic num at the start of the file
76 UWORD do_Version ; a version number, so we can change it
77 STRUCT do_Gadget,gg_SIZEOF ; a copy of in core gadget
78 UWORD do_Type
79 APTR do_DefaultTool
80 APTR do_ToolTypes
81 LONG do_CurrentX
82 LONG do_CurrentY
83 APTR do_DrawerData
84 APTR do_ToolWindow ; only applies to tools
85 LONG do_StackSize ; only applies to tools
86 LABEL do_SIZEOF
87
88 WB_DISKMAGIC EQU $e310 ; a magic number, not easily impersonated
89 WB_DISKVERSION EQU 1 ; our current version number
90
91 STRUCTURE FreeList,0
92 WORD fl_NumFree
93 STRUCT fl_MemList,LH_SIZE
94 ; weird name to avoid conflicts with FileLocks
95 LABEL FreeList_SIZEOF
96
97
98 STRUCTURE WBOBJECT,0
99 STRUCT wo_MasterNode,LN_SIZE ; all objects are on this list
100 STRUCT wo_Siblings,LN_SIZE ; list of drawer members
101 STRUCT wo_SelectNode,LN_SIZE ; list of all selected objects
102 STRUCT wo_UtilityNode,LN_SIZE ; function specific linkages
103 APTR wo_Parent
104
105 ; object flags -- see below for definitions
106 UBYTE wo_Flags
107
108 UBYTE wo_Type ; what flavor object is this?
109 USHORT wo_UseCount ; number of references to this obj
110 APTR wo_Name ; this object's textual name
111 SHORT wo_NameXoffset ; where to put the name
112 SHORT wo_NameYoffset
113
114 APTR wo_DefaultTool
115 APTR wo_DrawerData ; if this is a drawer or disk
116 APTR wo_IconWin ; each object's icon lives here
117 LONG wo_CurrentX ; virtual X in drawer
118 LONG wo_CurrentY ; virtual Y in drawer
119 APTR wo_ToolTypes ; the types for this tool

```

```

120 STRUCT wo_Gadget,gg_SIZEOF ; NOT a ptr, but an instance of it
121 STRUCT wo_FreeList,FreeList_SIZEOF ; this objects free list
122 APTR wo_ToolWindow ; character string for tool's window
123 LONG wo_StackSize ; how much stack to give to this
124 LONG wo_Lock ; if this tool is in the backdrop
125 LABEL wo_SIZEOF
126
127 ; workbench object flags
128 BITDEF WO_IconDisp,7 ; icon is currently in a window
129 BITDEF WO_DrawerOpen,6 ; we're a drawer, and it is open
130 BITDEF WO_Selected,5 ; our icon is selected
131 BITDEF WO_Background,4 ; set if icon is in background
132
133
134 * each message that comes into the WorkBenchPort must have a type field
135 * in the preceeding short. These are the defines for this type
136 *
137
138 MTYPE PSTD EQU 1 ; a "standard Potion" message
139 MTYPE_TOOLEXIT EQU 2 ; exit message from our tools
140 MTYPE_DISKCHANGE EQU 3 ; dos telling us of a disk change
141 MTYPE_TIMER EQU 4 ; we got a timer tick
142 MTYPE_CLOSEDOWN EQU 5 ; <unimplemented>
143 MTYPE_IOPROC EQU 6 ; <unimplemented>
144
145 ; we use the gadget id field to encode some special information
146 GID_WBOBJECT EQU 0 ; a normal workbench object
147 GID_HORIZSCROLL EQU 1 ; the horizontal scroll gadget for a drawer
148 GID_VERTSCROLL EQU 2 ; the vertical scroll gadget for a drawer
149 GID_LEFTSCROLL EQU 3 ; move one window left
150 GID_RIGHTSCROLL EQU 4 ; move one window right
151 GID_UPSCROLL EQU 5 ; move one window up
152 GID_DOWNSCROLL EQU 6 ; move one window down
153 GID_NAME EQU 7 ; the name field for an object
154
155
156 * workbench does different complement modes for its gadgets.
157 * It supports separate images, complement mode, and backfill mode.
158 * The first two are identical to intuitions GADGIMAGE and GADGHCOMP.
159 * backfill is similar to GADGHCOMP, but the region outside of the
160 * image (which normally would be color three when complemented)
161 * is flood-filled to color zero.
162 *
163 GADGBACKFILL EQU $0001
164
165 * if an icon does not really live anywhere, set its current position
166 * to here
167 *
168 NO_ICON_POSITION EQU ($80000000)

```

## File numbers for cross-reference listing:

|                   |                |                    |               |
|-------------------|----------------|--------------------|---------------|
| 1:adkbits.h       | 2:audio.h      | 3:blit.h           | 4:bootblock.h |
| 5:cia.h           | 6:clip.h       | 7:clipboard.h      | 8:collide.h   |
| 9:console.h       | 10:conunit.h   | 11:copper.h        | 12:ctype.h    |
| 13:custom.h       | 14:dec.h       | 15:disk.h          | 16:diskfont.h |
| 17:display.h      | 18:dmabits.h   | 19:dos.h           | 20:dosextns.h |
| 21:error.h        | 22:fcntl.h     | 23:gameport.h      | 24:gels.h     |
| 25:gfx.h          | 26:gfxbase.h   | 27:gfxmacros.h     | 28:graphint.h |
| 29:icon.h         | 30:input.h     | 31:inputevent.h    | 32:intbits.h  |
| 33:intuinternal.h | 34:intuition.h | 35:intuitionbase.h | 36:iosl.h     |
| 37:keyboard.h     | 38:keymap.h    | 39:layers.h        | 40:limits.h   |
| 41:macros.h       | 42:math.h      | 43:mathfp.h        | 44:misc.h     |
| 45:narrator.h     | 46:parallel.h  | 47:potgo.h         | 48:printer.h  |
| 49:prtbase.h      | 50:rastport.h  | 51:regions.h       | 52:serial.h   |
| 53:sprite.h       | 54:startup.h   | 55:stdio.h         | 56:text.h     |
| 57:timer.h        | 58:trackdisk.h | 59:translator.h    | 60:view.h     |
| 61:workbench.h    |                |                    |               |

|                       |                              |
|-----------------------|------------------------------|
| A_OR_B                | 3-34                         |
| A_OR_C                | 3-35                         |
| A_TO_D                | 3-37                         |
| A_XOR_C               | 3-36                         |
| ABC                   | 3-24, 3-34, 3-35, 3-37       |
| abms                  | 48-108                       |
| abnc                  | 3-25, 3-34, 3-35, 3-36, 3-37 |
| abs                   | 41-6, 43-31, 55-65           |
| ac_dat                | 13-91                        |
| ac_len                | 13-88                        |
| ac_pad                | 13-92                        |
| ac_per                | 13-89                        |
| ac_ptr                | 13-87                        |
| ac_vol                | 13-90                        |
| acam                  | 48-111                       |
| ACCESS_READ           | 19-57                        |
| ACCESS_WRITE          | 19-60                        |
| acos                  | 42-8, 42-94                  |
| ACTION_COPY_DIR       | 20-128                       |
| ACTION_CREATE_DIR     | 20-131                       |
| ACTION_CURRENT_VOLUME | 20-119                       |
| ACTION_DELETE_OBJECT  | 20-125                       |
| ACTION_DIE            | 20-117                       |
| ACTION_DISK_CHANGE    | 20-142                       |
| ACTION_DISK_INFO      | 20-134                       |
| ACTION_DISK_TYPE      | 20-141                       |
| ACTION_EVENT          | 20-118                       |
| ACTION_EXAMINE_NEXT   | 20-133                       |
| ACTION_EXAMINE_OBJECT | 20-132                       |
| ACTION_FREE_LOCK      | 20-124                       |
| ACTION_GET_BLOCK      | 20-115                       |
| ACTION_INFO           | 20-135                       |
| ACTION_INHIBIT        | 20-140                       |
| ACTION_LOCATE_OBJECT  | 20-120                       |
| ACTION_NIL            | 20-114                       |
| ACTION_PARENT         | 20-138                       |

|                       |                       |
|-----------------------|-----------------------|
| ACTION_READ           | 20-123                |
| ACTION_RENAME_DISK    | 20-121                |
| ACTION_RENAME_OBJECT  | 20-126                |
| ACTION_SET_COMMENT    | 20-137                |
| ACTION_SET_MAP        | 20-116                |
| ACTION_SET_PROTECT    | 20-130                |
| ACTION_TIMER          | 20-139                |
| ACTION_WAIT_CHAR      | 20-129                |
| ACTION_WRITE          | 20-122                |
| ACTIVATE              | 34-1287               |
| Activation            | 34-299                |
| ActiveGadget          | 33-123                |
| ActiveImage           | 33-125                |
| ActivePInfo           | 33-124                |
| ActiveScreen          | 33-79, 35-45          |
| ActiveWindow          | 33-78, 34-1005, 35-44 |
| ActiView              | 26-27                 |
| ADALLOC_MAXPREC       | 2-17                  |
| ADALLOC_MINPREC       | 2-16                  |
| ADCMD_ALLOCATE        | 2-27                  |
| ADCMD_FINISH          | 2-21                  |
| ADCMD_FREE            | 2-19                  |
| ADCMD_LOCK            | 2-23                  |
| ADCMD_PERVOL          | 2-22                  |
| ADCMD_SETPREC         | 2-20                  |
| ADCMD_WAITCYCLE       | 2-24                  |
| ADCMDB_NOUNIT         | 2-25                  |
| ADCMDF_NOUNIT         | 2-26, 2-27            |
| AddFreeList           | 29-40                 |
| aDEN1                 | 48-66                 |
| aDEN2                 | 48-65                 |
| aDEN3                 | 48-64                 |
| aDEN4                 | 48-63                 |
| aDENS                 | 48-62                 |
| aDEN6                 | 48-61                 |
| ADHARD_CHANNELS       | 2-14                  |
| ADIOB_NOWAIT          | 2-33                  |
| ADIOB_PERVOL          | 2-29                  |
| ADIOB_SYNC CYCLE      | 2-31                  |
| ADIOB_WRITE MESSAGE   | 2-35                  |
| ADIOERR_ALLOCFAILED   | 2-39                  |
| ADIOERR_CHANNELSTOLEN | 2-40                  |
| ADIOERR_NOALLOCATION  | 2-38                  |
| ADIOF_NOWAIT          | 2-34                  |
| ADIOF_PERVOL          | 2-30                  |
| ADIOF_SYNC CYCLE      | 2-32                  |
| ADIOF_WRITE MESSAGE   | 2-36                  |
| ADKB_FAST             | 1-21                  |
| ADKB_MEMPREC          | 1-17                  |
| ADKB_MSBSYNC          | 1-20                  |
| ADKB_PRECOMP0         | 1-16                  |
| ADKB_PRECOMP1         | 1-15                  |
| ADKB_SETCLR           | 1-14                  |
| ADKB_UARTBRK          | 1-18                  |
| ADKB_USEOP1           | 1-25                  |
| ADKB_USEOV1           | 1-29                  |

ADKB\_USE1P2, 1-24  
 ADKB\_USE1V2, 1-28  
 ADKB\_USE2P3, 1-23  
 ADKB\_USE2V3, 1-27  
 ADKB\_USE3PN, 1-22  
 ADKB\_USE3VN, 1-26  
 ADKB\_WORDSYNC, 1-19  
 ackconr, 13-85  
 ackconr, 13-29  
 ADKE\_FAST, 1-38  
 ADKE\_MEMPREC, 1-34  
 ADKE\_MSBSYNC, 1-37  
 ADKE\_PRE000NS, 1-48  
 ADKE\_PRE140NS, 1-49  
 ADKE\_PRE280NS, 1-50  
 ADKE\_PRE560NS, 1-51  
 ADKE\_PRECOMP0, 1-33, 1-49, 1-51  
 ADKE\_PRECOMP1, 1-32, 1-50, 1-51  
 ADKE\_SETCLR, 1-31  
 ADKE\_UARTBRK, 1-35  
 ADKE\_USE0P1, 1-42  
 ADKE\_USE0V1, 1-46  
 ADKE\_USE1P2, 1-41  
 ADKE\_USE1V2, 1-45  
 ADKE\_USE2P3, 1-40  
 ADKE\_USE2V3, 1-44  
 ADKE\_USE3PN, 1-39  
 ADKE\_USE3VN, 1-43  
 ADKE\_WORDSYNC, 1-36  
 aEXTEND, 48-121  
 af\_Attr, 16-67  
 af\_Type, 16-66  
 AEB\_DISK, 16-62  
 AEB\_MEMORY, 16-60  
 AEF\_DISK, 16-63  
 AEF\_MEMORY, 16-61  
 afh\_NumEntries, 16-71  
 aFNT0, 48-76  
 aFNT1, 48-77  
 aFNT10, 48-86  
 aFNT2, 48-78  
 aFNT3, 48-79  
 aFNT4, 48-80  
 aFNT5, 48-81  
 aFNT6, 48-82  
 aFNT7, 48-83  
 aFNT8, 48-84  
 aFNT9, 48-85  
 afp, 43-44  
 After, 24-160  
 AG\_OpenLib, 33-386, 33-387  
 ACNUS, 25-18, 25-19  
 aHTS, 48-113  
 aIND, 48-39  
 aJFY0, 48-95  
 aJFY1, 48-97

aJFY3, 48-96  
 aJFY5, 48-92  
 aJFY6, 48-94  
 aJFY7, 48-93  
 ALERT\_COUNTDOWN, 33-382  
 ALERT\_TYPE, 34-1889  
 ALERTLAYERSNOMEM, 39-43  
 AlgoStyle, 50-75  
 AllocWObject, 29-38  
 aLMS, 48-105  
 ALPHA\_P\_101, 34-1791  
 ALTKEYMAP, 34-493, 34-716  
 ALTLEFT, 34-1923  
 ALTRIGHT, 34-1924  
 AmigaIcon, 33-144  
 AMIGAKEYS, 34-1927  
 AMIGALEFT, 34-1925, 34-1927  
 AMIGARIGHT, 34-1926, 34-1927  
 AN\_Intuition, 33-386, 33-387  
 ANBC, 3-26, 3-34, 3-35, 3-37  
 ANBNC, 3-27, 3-34, 3-35, 3-36, 3-37  
 aNEL, 48-40  
 ANFRACSIZE, 24-47  
 AnimBob, 24-203  
 AnimComp, 24-164, 24-171, 24-189, 24-190, 24-193,  
 24-194, 24-228  
 AnimCRoutine, 24-196  
 ANIMHALF, 24-48  
 animKey, 24-253, 24-253  
 AnimOb, 24-201, 24-206, 24-209  
 AnimORoutine, 24-225  
 AnOldX, 24-214  
 AnOldY, 24-214  
 AnX, 24-217  
 AnY, 24-217  
 AO\_GraphicsLib, 33-386  
 AO\_LayersLib, 33-387  
 AOLPen, 27-28, 50-63  
 apattern, 33-146  
 APen, 33-236  
 aPERF, 48-102  
 aPERFO, 48-103  
 aPLD, 48-74  
 aPLU, 48-73  
 APointer, 33-99  
 aPROPO, 48-90  
 aPROPL, 48-89  
 aPROP2, 48-88  
 APtrHeight, 33-101  
 APtrWidth, 33-102  
 AreaInfo, 50-18, 50-58, 50-58  
 AREAOUTLINE, 27-28, 27-33, 50-101  
 AreaPtrn, 27-31, 50-56  
 AreaPtSz, 27-31, 50-65  
 arg1, 42-38  
 arg2, 42-38

aRI, 48-41  
 aRIN, 48-38  
 aRIS, 48-37  
 aRMS, 48-106  
 aSBC, 48-51  
 aSFC, 48-50  
 aSGR0, 48-43  
 aSGR1, 48-48  
 aSGR22, 48-49  
 aSGR23, 48-45  
 aSGR24, 48-47  
 aSGR3, 48-44  
 aSGR4, 48-46  
 ASHIFTSHIFT, 3-55  
 aSHORP0, 48-53  
 aSHORP1, 48-55  
 aSHORP2, 48-54  
 aSHORP3, 48-57  
 aSHORP4, 48-56  
 aSHORP5, 48-59  
 aSHORP6, 48-58  
 asin, 42-9, 42-94  
 aSLPP, 48-101  
 aSLRM, 48-110  
 ASPECT\_HORIZ, 34-1774  
 ASPECT\_VERT, 34-1775  
 aSTBM, 48-109  
 aSUS0, 48-72  
 aSUS1, 48-69  
 aSUS2, 48-68  
 aSUS3, 48-71  
 aSUS4, 48-70  
 AT\_DeadEnd, 33-386, 33-387  
 atan, 42-10, 42-95  
 atan2, 42-95  
 aTBC0, 48-115  
 aTBC1, 48-117  
 aTBC3, 48-116  
 aTBC4, 48-118  
 aTBCALL, 48-119  
 aTBSALL, 48-120  
 aTMS, 48-107  
 atof, 42-92  
 atoi, 42-90  
 atol, 42-91  
 aTSS, 48-91  
 aud, 13-93  
 AudChannel, 13-86  
 audio, 2-12  
 AUDIONAME, 2-12  
 AUL, 3-70  
 AUserExt, 24-230  
 AUserStuff, 24-66, 24-67, 24-230  
 AUTOBACKPEN, 34-1905  
 AUTODRAWMODE, 34-1906  
 AUTOFRONTPEN, 34-1904

AUTOITEXTEFONT, 34-1909  
 AUTOKNOB, 34-629  
 AUTOLEFTEDGE, 34-1907  
 AUTONEXTTEXT, 34-1910  
 AUTOTOPEDGE, 34-1908  
 AvailFonts, 16-65  
 AvailFontsHeader, 16-70  
 aVERP0, 48-99  
 aVERP1, 48-100  
 aVTS, 48-114  
 AXOffset, 33-104  
 AYOffset, 33-104  
 B2BOBER, 24-261  
 B2NORM, 24-259  
 B2SWAP, 24-260  
 back, 6-27  
 BACKDROP, 34-1271  
 BackFill, 34-229  
 BackPen, 34-738, 34-787  
 BACKSAVED, 24-28  
 BADDR, 19-119  
 BADGDCGET, 33-389  
 BADMESSAGE, 33-391  
 BADSTATE, 33-390  
 BarHBorder, 33-162, 34-1468  
 BarHeight, 34-1468  
 BarLayer, 34-1506  
 BarVBorder, 33-161, 34-1468  
 BAUD\_110, 34-1739  
 BAUD\_1200, 34-1741  
 BAUD\_19200, 34-1745  
 BAUD\_2400, 34-1742  
 BAUD\_300, 34-1740  
 BAUD\_4800, 34-1743  
 BAUD\_9600, 34-1744  
 BAUD\_MIDI, 34-1746  
 BaudRate, 34-1643  
 bb\_chksum, 4-21  
 bb\_dosblock, 4-22  
 bb\_id, 4-20  
 BBID\_DOS, 4-27  
 BBID\_KICK, 4-28  
 BBNAME\_DOS, 4-30  
 BBNAME\_KICK, 4-31  
 BCOB\_DEST, 3-39  
 BCOB\_SRCA, 3-42  
 BCOB\_SRCB, 3-41  
 BCOB\_SRCC, 3-40  
 BCOE\_DEST, 3-43  
 BCOE\_SRCA, 3-46  
 BCOE\_SRCB, 3-45  
 BCOE\_SRCC, 3-44  
 BCLF\_DESC, 3-48  
 BDRAWN, 24-40  
 beamsync, 3-88, 26-41  
 BeatX, 34-87



BeatY, 34-87  
 BEEPING, 34-1534  
 Before, 24-159  
 BgPen, 50-62  
 BITCLR, 25-16, 27-21, 27-23, 27-26  
 BitMap, 6-39, 6-64, 6-64, 25-33, 33-114, 33-115,  
 34-245, 34-1392, 34-1392, 34-1480, 34-1480,  
 34-1587, 50-55, 50-55, 60-70, 60-70  
 BITSET, 25-15, 27-20, 27-22, 27-25  
 BITSPEERBYTE, 19-44  
 BITSPEERLONG, 19-46  
 BlitLock, 26-48  
 BLITMSG\_FAULT, 26-63  
 BlitNest, 26-49  
 BlitOwner, 26-52  
 BLITREVERSE, 3-66  
 blitsize, 3-87  
 blitter, 26-30  
 BlitWaitQ, 26-51  
 blockpen, 33-139, 34-1193, 34-1342, 34-1496, 34-1558  
 bltadat, 13-69  
 bltafwm, 13-54  
 bltalwm, 13-55  
 bltamod, 13-64  
 bltapt, 13-58  
 bltbdat, 13-68  
 bltbmod, 13-63  
 bltbpt, 13-57  
 bltcdat, 13-67  
 bltcmmod, 13-62  
 bltcon0, 13-52  
 bltcon1, 13-53  
 bltcpt, 13-56  
 bltddat, 13-21  
 bltdmod, 13-65  
 bltdpt, 13-59  
 blthd, 26-33  
 bltnode, 3-82, 3-84, 26-33, 26-34  
 bltsize, 13-60  
 bltsrv, 26-35  
 blttl, 26-33  
 BNDRYOFF, 27-33  
 Bob, 24-122, 24-142, 24-159, 24-160, 24-203  
 BobComp, 24-164  
 BOBISCOMP, 24-37  
 BOBNIX, 24-42  
 BOBSAWAY, 24-41, 24-254  
 BOBUPDATE, 24-29  
 BobVSprite, 24-162  
 BOOLGADGET, 34-526  
 BootBlock, 4-19  
 BOOTSECTS, 4-25  
 Border, 34-222, 34-783, 34-796  
 BorderBottom, 34-1154  
 BORDERHIT, 8-24  
 BorderLeft, 34-1154

BORDERLESS, 34-1284  
 BorderLine, 24-116  
 BorderRight, 34-1154  
 BorderRPort, 34-1155  
 BorderTop, 34-1154  
 BOTTOMBORDER, 34-476  
 BOTTOMHIT, 8-34  
 bottommost, 50-48  
 bounds, 6-30, 6-65, 51-15, 51-20  
 bpattern, 33-146  
 BPen, 33-236  
 bpl1mod, 13-100  
 bpl2mod, 13-101  
 bplcon0, 13-96  
 bplcon1, 13-97  
 bplcon2, 13-98  
 bpldat, 13-103  
 bplpt, 13-94  
 broadcast, 39-30  
 BROTHER\_15XL, 34-1792  
 bsblthd, 26-34  
 bsblttl, 26-34  
 BSHIFTSHIFT, 3-56  
 BufBuffer, 24-243  
 Buffer, 34-670  
 BufferPos, 34-674  
 BufPath, 24-239  
 BUFSIZ, 55-8  
 BuFX, 24-238  
 BuFY, 24-238  
 BUserExt, 24-168  
 BUSERFLAGS, 24-35  
 BUserStuff, 24-62, 24-63, 24-168  
 BWAITING, 24-39  
 bytereserved, 26-45, 39-35  
 BYTESPEERLONG, 19-45  
 BytesPerRow, 25-35  
 Carg, 28-19  
 CBD\_CURRENTREADID, 7-24  
 CBD\_CURRENTWRITEID, 7-25  
 CBD\_POST, 7-23  
 CBERR\_OBSOLETEID, 7-27  
 CBM\_MPS1000, 34-1793  
 CBump, 27-36, 27-37  
 ccode, 28-18  
 CD\_ASKKEYMAP, 9-24  
 CD\_SETKEYMAP, 9-25  
 cell, 42-93  
 CEND, 27-38  
 ch\_masks, 45-68  
 chanmask, 45-73  
 check\_ip, 39-25  
 CHECKED, 34-183  
 CheckImage, 33-144  
 CHECKIT, 34-155  
 CheckMark, 34-1203, 34-1367

CHECKWIDTH, 34-1877  
 Height, 34-619  
 cia, 26-29  
 ctaa, 5-11  
 CIAANAME, 5-11  
 ciab, 5-12  
 CIABNAME, 5-12  
 CINIT, 27-35  
 Class, 34-921  
 CLEANME, 3-94  
 cleanup, 3-89, 3-93, 3-94  
 clearerr, 55-58  
 ClearPath, 24-87  
 cleft, 33-140, 34-689  
 cli\_Background, 20-193  
 cli\_CommandDir, 20-184  
 cli\_CommandFile, 20-191  
 cli\_CommandName, 20-186  
 cli\_CurrentInput, 20-190  
 cli\_CurrentOutput, 20-194  
 cli\_DefaultStack, 20-195  
 cli\_FailLevel, 20-187  
 cli\_Interactive, 20-192  
 cli\_Module, 20-197  
 cli\_Prompt, 20-188  
 cli\_Result2, 20-182  
 cli\_ReturnCode, 20-185  
 cli\_SetName, 20-183  
 cli\_StandardInput, 20-189  
 cli\_StandardOutput, 20-196  
 clip, 34-29  
 ClipboardUnitPartial, 7-30  
 ClipRect, 6-28, 6-28, 6-40, 6-51, 6-54, 6-55, 6-59,  
 6-61, 6-62, 6-66, 33-112, 33-113  
 Clock, 24-212  
 CLOSE, 34-524  
 CLOSEGADCET, 33-61  
 CLOSEWINDOW, 34-987  
 clrerr, 55-58  
 ClrIns, 60-56  
 clxcon, 13-82  
 clxdat, 13-28  
 CMD\_CLEAR, 58-101  
 CMD\_READ, 58-96  
 CMD\_UPDATE, 58-100  
 CMD\_WRITE, 58-95  
 CMOVE, 27-36, 27-36  
 code, 28-17, 34-925  
 collHandler, 50-47  
 CollMask, 24-117  
 collPtrs, 24-268  
 collTable, 24-266, 50-47  
 color, 13-112  
 color0, 33-166, 33-265, 34-1673  
 color1, 33-167, 33-266, 34-1674  
 color17, 33-170, 33-259, 34-1665

color18, 33-171, 33-260, 34-1666  
 color19, 33-172, 33-261, 34-1668  
 color2, 33-168, 33-267, 34-1676  
 color3, 33-169, 33-268, 34-1678  
 ColorMap, 48-145, 60-20, 60-32, 60-32  
 COLORON, 17-19  
 ColorTable, 60-25  
 COM\_MENU, 33-314  
 COM\_SELECT, 33-313  
 Command, 34-140  
 CommandLineInterface, 20-181  
 COMMSEQ, 34-163  
 COMMWIDTH, 34-1878  
 COMPLEMENT, 50-90  
 console, 10-16  
 ConUnit, 10-32  
 cop1lc, 13-72  
 cop2lc, 13-73  
 copcon, 13-43  
 copinit, 11-78, 26-28, 26-28  
 CopIns, 11-20, 11-62, 11-62, 11-63, 13-76  
 copjmpl, 13-74  
 copjmp2, 13-75  
 CopList, 11-25, 11-57, 11-59, 11-60, 11-74, 11-75,  
 11-75, 60-34, 60-35, 60-36  
 CopLStart, 11-64  
 COPPER\_MOVE, 11-15  
 COPPER\_WAIT, 11-16  
 CopPtr, 11-63  
 CopSStart, 11-65  
 cos, 42-11, 42-94  
 cosh, 42-12, 42-94  
 cot, 42-13, 42-94  
 Count, 11-66, 34-791, 50-24, 60-24  
 cp\_x, 50-70  
 cp\_y, 50-70  
 CPR\_NT\_LOF, 11-18  
 CPR\_NT\_SHT, 11-19  
 cprlist, 11-50, 11-52, 60-48, 60-49  
 CPRNXTBUF, 11-17  
 cr, 6-55  
 cr2, 6-55  
 CR\_NEEDS\_NO\_CONCEALED\_RASTERS, 6-74  
 crnew, 6-55  
 CTC\_HCLRTAB, 9-84  
 CTC\_HCLRTABSALL, 9-85  
 CTC\_HSETTAB, 9-83  
 ctl, 13-108  
 ctop, 33-140, 34-689  
 cu\_AlgoStyle, 10-67  
 cu\_AOLPen, 10-61  
 cu\_AreaPtrn, 10-64  
 cu\_AreaPtSz, 10-63  
 cu\_BgPen, 10-60  
 cu\_DrawMode, 10-62  
 cu\_FgPen, 10-59

cu\_Font, 10-66  
 cu\_KeyMapStruct, 10-53  
 cu\_Mask, 10-58  
 cu\_MinTerms, 10-65  
 cu\_Modes, 10-75  
 cu\_MP, 10-33  
 cu\_Node, 7-31  
 cu\_RawEvents, 10-76  
 cu\_TabStops, 10-55  
 cu\_TxBaseline, 10-71  
 cu\_TxFlags, 10-68  
 cu\_TxHeight, 10-69  
 cu\_TxSpacing, 10-72  
 cu\_TxWidth, 10-70  
 cu\_UnitNum, 7-32  
 cu\_Window, 10-35  
 cu\_XCCP, 10-48  
 cu\_XCP, 10-36  
 cu\_XMax, 10-38  
 cu\_XMinShrink, 10-46  
 cu\_XRExtant, 10-44  
 cu\_XROrigin, 10-42  
 cu\_XRSize, 10-40  
 cu\_YCCP, 10-49  
 cu\_YCP, 10-37  
 cu\_YMax, 10-39  
 cu\_YMinShrink, 10-47  
 cu\_YRExtant, 10-45  
 cu\_YROrigin, 10-43  
 cu\_YRSize, 10-41  
 CURSORDOWN, 34-1932  
 CursorDX, 33-185  
 CursorDY, 33-185  
 CURSORLEFT, 34-1930  
 CURSORRIGHT, 34-1931  
 CURSORUP, 34-1929  
 Custom, 13-20, 27-20, 27-21, 27-22, 27-23, 27-25,  
 27-26, 34-1787  
 CUSTOM\_NAME, 34-1790  
 CUSTOMBITMAP, 34-1537, 34-1587  
 CUSTOMSCREEN, 34-1527  
 cvfd, 14-34  
 cvfdx, 14-34  
 CWAIT, 27-37, 27-37, 27-38  
 Cwidth, 34-617  
 D, 4-27, 4-30, 19-156, 19-157, 19-158  
 D005, 14-20  
 D05, 14-20  
 D5, 14-20  
 D\_AUX, 36-35  
 D\_CON, 36-33  
 D\_DIC, 14-16, 14-17  
 D\_DISK, 36-32  
 D\_MAX, 14-17  
 D\_NULL, 36-36  
 D\_PRN, 36-34

DamageList, 6-49  
 dataa, 13-109  
 datab, 13-110  
 DateStamp, 19-62, 19-89, 20-165, 20-211  
 dbf, 43-44  
 DBLFF, 17-20  
 DBuffer, 24-166, 50-96  
 DBufPacket, 24-166, 24-236  
 dd\_Children, 61-62  
 dd\_CmdBytes, 49-53  
 dd\_CmdVectors, 49-52  
 dd\_CurrentX, 61-44  
 dd\_CurrentY, 61-45  
 dd\_Device, 49-49  
 dd\_DownMove, 61-53  
 dd\_DrawerWin, 61-60  
 dd\_ExecBase, 49-51  
 dd\_HorizImage, 61-56  
 dd\_HorizProp, 61-58  
 dd\_HorizScroll, 61-50  
 dd\_LeftMove, 61-54  
 dd\_Lock, 61-63  
 dd\_MaxX, 61-48  
 dd\_MaxY, 61-49  
 dd\_MinX, 61-46  
 dd\_MinY, 61-47  
 dd\_NewWindow, 61-43  
 dd\_NumCommands, 49-54  
 dd\_Object, 61-61  
 dd\_RightMove, 61-55  
 dd\_Segment, 49-50  
 dd\_UpMove, 61-52  
 dd\_VertImage, 61-57  
 dd\_VertProp, 61-59  
 dd\_VertScroll, 61-51  
 ddfstop, 13-80  
 ddfstrt, 13-79  
 DEADEND\_ALERT, 34-1892  
 Debug, 26-40  
 DefaultFont, 26-37  
 DefaultTitle, 34-1463, 34-1570  
 DEFERRED, 33-312  
 DEFERREFRESH, 34-272  
 DEFFREQ, 45-37  
 DEFFMODE, 45-43  
 DEFPITCH, 45-34  
 DEFERATE, 45-35  
 DEFSEX, 45-42  
 DEFVOL, 45-36  
 DELTAMOVE, 34-1009  
 Depth, 24-106, 25-38, 34-827, 34-1555  
 Descendant, 34-1168  
 DEST, 3-50  
 DestAddr, 11-31, 11-44, 11-44  
 DestData, 11-36, 11-46, 11-46  
 detailpen, 33-139, 34-1193, 34-1342, 34-1496, 34-1558

device, 2-12, 7-39, 46-85, 48-125, 48-139, 52-147,  
 57-26, 58-73  
 DeviceData, 49-48, 49-61  
 DeviceList, 20-206  
 devices, 10-16, 10-20, 10-24, 31-14, 34-57, 34-61,  
 49-32, 49-35, 49-38  
 DEVICES\_AUDIO\_H, 2-1, 2-2  
 DEVICES\_CLIPBOARD\_H, 7-1, 7-2  
 DEVICES\_CONSOLE\_H, 9-1, 9-2, 10-15  
 DEVICES\_GAMEPORT\_H, 23-1, 23-2  
 DEVICES\_INPUT\_H, 30-1, 30-2  
 DEVICES\_INPUTEVENT\_H, 10-23, 31-1, 31-2, 34-60  
 DEVICES\_KEYBOARD\_H, 37-1, 37-2  
 DEVICES\_KEYMAP\_H, 10-19, 38-1, 38-2  
 DEVICES\_NARRATOR\_H, 45-1, 45-2, 45-92  
 DEVICES\_PARALLEL\_H, 46-18, 46-19, 46-98, 49-31  
 DEVICES\_PRINTER\_H, 48-1, 48-2  
 DEVICES\_PRTBASE\_H, 49-11, 49-12  
 DEVICES\_SERIAL\_H, 49-34, 52-16, 52-17, 52-149  
 DEVICES\_TIMER\_H, 31-13, 34-52, 49-37, 57-15, 57-16, 57-43  
 DEVICES\_TRACKDISK\_H, 58-19, 58-20, 58-143  
 dfh\_DF, 16-51  
 dfh\_FileID, 16-52  
 DFH\_ID, 16-40  
 dfh\_Name, 16-55  
 dfh\_Revision, 16-53  
 dfh\_Segment, 16-54  
 dfh\_TF, 16-56  
 DETCH\_MASK, 17-35  
 DHeight, 60-38  
 di\_Devices, 20-173  
 di\_DevInfo, 20-172  
 di\_Handlers, 20-174  
 di\_McName, 20-171  
 di\_NetHand, 20-175  
 DIAB\_630, 34-1794  
 DIAB\_ADV\_D25, 34-1795  
 DIAB\_C\_150, 34-1796  
 diagstrt, 11-80  
 DiscResource, 15-56  
 DiscResourceUnit, 15-49, 15-58  
 disk, 15-107  
 DiskFontHeader, 16-43  
 DISKINSERTED, 34-999  
 DISKNAME, 15-107  
 DiskObject, 61-70  
 DISKREMOVED, 34-1001  
 DispCount, 34-687  
 DisplayFlags, 26-54  
 DispPos, 34-678  
 DISTANTACTIVE, 33-217  
 DistantEcho, 33-181, 33-206, 33-208  
 DISTANTMOVEWINDOW, 33-221  
 DISTANTNEWPREFS, 33-220  
 DISTANTPATROL, 33-218  
 DISTANTREQCLEAR, 33-226

DISTANTREQSET, 33-225  
 DistantScreen, 33-210  
 DISTANTSCREENBAR, 33-219  
 DISTANTSIZEWINDOW, 33-222  
 DistantWindow, 33-209  
 DISTANTWINDOWBACK, 33-224  
 DISTANTWINDOWERONT, 33-223  
 DIW\_HORIZ\_POS, 17-30  
 DIW\_VRTCL\_POS, 17-31  
 DIW\_VRTCL\_POS\_SHIFT, 17-32  
 diwstop, 13-78  
 diwstrt, 13-77  
 dl\_A2, 20-153  
 dl\_A5, 20-154  
 dl\_A6, 20-155  
 dl\_DiskType, 20-213  
 dl\_GV, 20-152  
 dl\_Lib, 20-150  
 dl\_Lock, 20-210  
 dl\_LockList, 20-212  
 dl\_Name, 20-215  
 dl\_Next, 20-207  
 dl\_Root, 20-151  
 dl\_Task, 20-209  
 dl\_Type, 20-208  
 dl\_unused, 20-214  
 dl\_VolumeDate, 20-211  
 DLT\_DEVICE, 20-219  
 DLT\_DIRECTORY, 20-220  
 DLT\_VOLUME, 20-221  
 DMAB\_AUDIO, 18-38  
 DMAB\_AUD1, 18-39  
 DMAB\_AUD2, 18-40  
 DMAB\_AUD3, 18-41  
 DMAB\_BLITHOC, 18-48  
 DMAB\_BLITTER, 18-44  
 DMAB\_BLTDONE, 18-49  
 DMAB\_BLTNZERO, 18-50  
 DMAB\_COPPER, 18-45  
 DMAB\_DISK, 18-42  
 DMAB\_MASTER, 18-47  
 DMAB\_RASTER, 18-46  
 DMAB\_SETCLR, 18-37  
 DMAB\_SPRITE, 18-43  
 dmacon, 13-81, 27-20, 27-21, 27-22, 27-23  
 dmaconr, 13-22  
 DMAE\_ALL, 18-30  
 DMAE\_AUDIO, 18-19  
 DMAE\_AUD1, 18-20  
 DMAE\_AUD2, 18-21  
 DMAE\_AUD3, 18-22  
 DMAE\_AUDIO, 18-18  
 DMAE\_BLITHOC, 18-29  
 DMAE\_BLITTER, 18-25  
 DMAE\_BLTDONE, 18-34  
 DMAE\_BLTNZERO, 18-35

DMAE\_COPPER, 18-26  
 DMAE\_DISK, 18-23  
 DMAE\_MASTER, 18-28  
 DMAE\_RASTER, 18-27, 27-20, 27-21  
 DMAE\_SETCLR, 18-17  
 DMAE\_SPRITE, 18-24, 27-22, 27-23  
 DMODECOUNT, 33-46, 33-156, 33-157, 33-158, 33-159,  
 33-161, 33-162, 33-163, 33-164  
 DMRequest, 34-1125  
 do\_CurrentX, 61-77  
 do\_CurrentY, 61-78  
 do\_DefaultTool, 61-75  
 do\_DrawerData, 61-79  
 do\_Gadget, 61-73  
 do\_Magic, 61-71  
 do\_StackSize, 61-81  
 do\_ToolTypes, 61-76  
 do\_ToolWindow, 61-80  
 do\_Type, 61-74  
 do\_Version, 61-72  
 DOMAIN, 42-47  
 dos, 19-19, 19-53, 19-106, 19-160, 19-213, 20-24,  
 21-8, 54-18  
 dosextens, 49-41  
 DosInfo, 20-170  
 DosLibrary, 20-149  
 DOSNAME, 19-19  
 DosPacket, 20-77, 20-110  
 DoubleClick, 34-1651  
 DoubleMicros, 33-154  
 DoubleSeconds, 33-154  
 DOWNBACKGADGET, 33-59  
 dp\_Action, 20-91  
 dp\_Arg1, 20-94, 20-95  
 dp\_Arg2, 20-96  
 dp\_Arg3, 20-97  
 dp\_Arg4, 20-98  
 dp\_Arg5, 20-99  
 dp\_Arg6, 20-100  
 dp\_Arg7, 20-101  
 dp\_BufAddr, 20-94  
 dp\_Link, 20-78  
 dp\_Port, 20-79  
 dp\_Res1, 20-84, 20-92  
 dp\_Res2, 20-88, 20-93  
 dp\_Status, 20-92  
 dp\_Status2, 20-93  
 dp\_Type, 20-81, 20-91  
 DPR, 14-24  
 DR\_ALLOCUNIT, 15-110  
 dr\_CiaResource, 15-62  
 dr\_Current, 15-58  
 dr\_DiscBlock, 15-65  
 dr\_DiscSync, 15-66  
 dr\_Flags, 15-59  
 DR\_FREEUNIT, 15-111

DR\_GETUNIT, 15-112  
 DR\_GETUNITID, 15-114  
 DR\_GIVEUNIT, 15-113, 15-117  
 dr\_Index, 15-67  
 DR\_LASTCOMM, 15-117  
 dr\_Library, 15-57  
 dr\_pad, 15-60  
 dr\_SysLib, 15-61  
 dr\_UnitID, 15-63  
 dr\_Waiting, 15-64  
 DRAFT, 33-282, 34-1758  
 DRAGGADGET, 33-62  
 DRAGSELECT, 33-309  
 drand48, 42-96  
 DrawerData, 61-42, 61-79, 61-119  
 DRAWERDATAFILESIZE, 61-67  
 DrawMode, 34-740, 34-789, 50-64  
 DrawPath, 24-86  
 DRB\_ACTIVE, 15-75  
 DRB\_ALLOC0, 15-71  
 DRB\_ALLOC1, 15-72  
 DRB\_ALLOC2, 15-73  
 DRB\_ALLOC3, 15-74  
 DRE\_ACTIVE, 15-81  
 DRE\_ALLOC0, 15-77  
 DRE\_ALLOC1, 15-78  
 DRE\_ALLOC2, 15-79  
 DRE\_ALLOC3, 15-80  
 DRT\_37422D2S, 15-126  
 DRT\_AMIGA, 15-125  
 DRT\_EMPTY, 15-127  
 dru\_DiscBlock, 15-51  
 dru\_DiscSync, 15-52  
 dru\_Index, 15-53  
 dru\_Message, 15-50  
 ds\_Days, 19-63  
 ds\_Minute, 19-65  
 ds\_Tick, 19-67  
 dskbytr, 13-34  
 dskdat, 13-39  
 dskdatr, 13-25  
 DSKDMAOFF, 15-92  
 dsklen, 13-38  
 dskpt, 13-37  
 dsksync, 13-71  
 DepIns, 60-34  
 DSR\_CPR, 9-80  
 DUALPF, 60-57  
 dummy, 50-67  
 DWidth, 60-38  
 DxOffset, 60-39, 60-50  
 DyOffset, 11-68, 60-39, 60-50  
 E, 14-22, 43-15  
 Echo, 33-208  
 Echoes, 33-181  
 ecvt, 42-88

EIGHT\_LPI, 34-1767  
 ELITE, 34-1754  
 else, 25-21, 55-38, 61-106  
 ENABLECLI, 33-275, 34-1688  
 ENDGADGET, 34-446  
 EOF, 55-43  
 EPSON, 33-278, 34-1797  
 EPSON\_JX\_80, 34-1798  
 erand48, 42-96  
 errno, 42-86  
 ERROR\_ACTION\_NOT\_KNOWN, 19-176  
 ERROR\_BAD\_STREAM\_NAME, 19-174  
 ERROR\_COMMENT\_TOO\_BIG, 19-187  
 ERROR\_DELETE\_PROTECTED, 19-189  
 ERROR\_DEVICE\_NOT\_MOUNTED, 19-185  
 ERROR\_DIR\_NOT\_FOUND, 19-172  
 ERROR\_DIRECTORY\_NOT\_EMPTY, 19-183  
 ERROR\_DISK\_FULL, 19-188  
 ERROR\_DISK\_NOT\_VALIDATED, 19-180  
 ERROR\_DISK\_WRITE\_PROTECTED, 19-181  
 ERROR\_INVALID\_COMPONENT\_NAME, 19-177  
 ERROR\_INVALID\_LOCK, 19-178  
 ERROR\_NO\_DEFAULT\_DIR, 19-169  
 ERROR\_NO\_DISK, 19-193  
 ERROR\_NO\_FREE\_STORE, 19-168  
 ERROR\_NO\_MORE\_ENTRIES, 19-194  
 ERROR\_NOT\_A\_DOS\_DISK, 19-192  
 ERROR\_OBJECT\_EXISTS, 19-171  
 ERROR\_OBJECT\_IN\_USE, 19-170  
 ERROR\_OBJECT\_NOT\_FOUND, 19-173  
 ERROR\_OBJECT\_TOO\_LARGE, 19-175  
 ERROR\_OBJECT\_WRONG\_TYPE, 19-179  
 ERROR\_READ\_PROTECTED, 19-191  
 ERROR\_RENAME\_ACROSS\_DEVICES, 19-182  
 ERROR\_SEEK\_ERROR, 19-186  
 ERROR\_TOO\_MANY\_LEVELS, 19-184  
 ERROR\_WRITE\_PROTECTED, 19-190  
 ErrorX, 33-191  
 ErrorY, 33-191  
 ETD\_CLEAR, 58-101  
 ETD\_FORMAT, 58-99  
 ETD\_MOTOR, 58-97  
 ETD\_READ, 58-96  
 ETD\_SEEK, 58-98  
 ETD\_UPDATE, 58-100  
 ETD\_WRITE, 58-95  
 EventCount, 33-120  
 EVENTMAX, 33-50, 33-121  
 except, 42-95  
 exception, 42-34  
 EXCLUSIVE\_LOCK, 19-58  
 ExecMessage, 34-915  
 exp, 42-14, 42-92  
 ExtData, 34-1231, 34-1508  
 EXTRA\_HALFBRITE, 60-65  
 F\_DUPED, 22-23

F\_GETED, 22-24  
 F\_GETEL, 22-26  
 F\_SETED, 22-25  
 F\_SETEL, 22-27  
 fabs, 42-15, 42-93  
 FALSE, 33-275  
 FANFOLD, 33-295, 34-1749  
 fatten\_count, 39-34  
 fc\_FileName, 16-26  
 fc\_Flags, 16-29  
 fc\_Style, 16-28  
 fc\_Size, 16-27  
 fch\_FileID, 16-35  
 FCH\_ID, 16-32  
 fch\_NumEntries, 16-36  
 FDEDCP, 14-31  
 FDEDIT, 14-29  
 EDMONY, 14-32  
 EDTYPE, 14-30  
 FEMALE, 45-39  
 feof, 55-53  
 ferror, 55-54  
 fetchIBase, 33-411  
 fflush, 55-57  
 fgets, 55-63  
 FgPen, 50-61  
 fh\_Arg1, 20-71  
 fh\_Arg2, 20-72  
 fh\_Args, 20-70, 20-71  
 fh\_Buf, 20-63  
 fh\_End, 20-65  
 fh\_Func1, 20-67  
 fh\_Func2, 20-68  
 fh\_Func3, 20-69  
 fh\_Funcs, 20-66, 20-67  
 fh\_Link, 20-59  
 fh\_Port, 20-60  
 fh\_Pos, 20-64  
 fh\_Type, 20-61  
 fib\_Comment, 19-90  
 fib\_Date, 19-89  
 fib\_DirEntryType, 19-77  
 fib\_DiskKey, 19-76  
 fib\_EntryType, 19-84  
 fib\_FileName, 19-80  
 fib\_NumBlocks, 19-87  
 fib\_Protection, 19-82  
 fib\_Size, 19-85  
 FIBB\_DELETE, 19-101, 19-105  
 FIBB\_EXECUTE, 19-100, 19-104  
 FIBB\_READ, 19-98, 19-102  
 FIBB\_WRITE, 19-99, 19-103  
 FIBF\_DELETE, 19-105  
 FIBF\_EXECUTE, 19-104  
 FIBF\_READ, 19-102  
 FIBF\_WRITE, 19-103

FileHandle, 20-58  
 FileInfoBlock, 19-75  
 FileLock, 20-225  
 FILENAME\_SIZE, 34-1606, 34-1694  
 fileno, 55-55  
 FILL\_CARRYIN, 3-62  
 FILL\_OR, 3-60  
 FILL\_XOR, 3-61  
 FINE, 34-1755  
 firstBlissObj, 50-49  
 FirstCopList, 11-74  
 FirstGadget, 34-1165, 34-1358, 34-1490  
 FirstItem, 34-82  
 FirstRequest, 34-1118  
 FirstScreen, 33-84, 35-50  
 FirstWindow, 34-1447  
 firstx, 33-137, 50-26  
 firsty, 33-137, 50-26  
 fl\_Access, 20-228  
 fl\_Key, 20-227  
 fl\_Link, 20-226  
 fl\_MemList, 61-90  
 fl\_NumFree, 61-89  
 fl\_Task, 20-229  
 fl\_Volume, 20-230  
 FlagPtr, 50-23  
 Flags, 6-38, 6-69, 24-95, 24-148, 24-176, 24-254,  
 25-37, 26-47, 27-28, 27-29, 27-33, 33-86,  
 33-211, 34-78, 34-122, 34-225, 34-296,  
 34-553, 34-1109, 34-1348, 34-1458, 39-32,  
 50-41, 50-68, 60-22  
 FlagTbl, 50-22  
 floor, 42-93  
 fmod, 42-93  
 FOLLOWMOUSE, 34-465  
 font, 33-422, 34-1472, 34-1567, 50-74  
 FontContents, 16-25  
 FontContentsHeader, 16-34  
 FontHeight, 34-1634  
 fopen, 55-60  
 for, 34-1863  
 FOREVER, 34-1863  
 fp, 55-56, 55-56, 55-57, 55-57, 55-58, 55-58  
 FPB\_DESIGNED, 56-40  
 FPB\_DISKFONT, 56-30  
 FPB\_PROPORTIONAL, 56-38  
 FPB\_REMOVED, 56-42  
 FPB\_REVPATH, 56-32  
 FPB\_ROMFONT, 56-28  
 FPB\_TALLDOT, 56-34  
 FPB\_WIDEDOT, 56-36  
 FPENAN, 42-62  
 FPEOVE, 42-60  
 FPEUND, 42-59  
 FPEZDV, 42-61  
 FPF\_DESIGNED, 56-41

FPF\_DISKFONT, 56-31  
 FPF\_PROPORTIONAL, 56-39  
 FPF\_REMOVED, 56-43  
 FPF\_REVPATH, 56-33  
 FPF\_ROMFONT, 56-29  
 FPF\_TALLDOT, 56-35  
 FPF\_WIDEDOT, 56-37  
 FPHALF, 43-20  
 FPONE, 43-19  
 EPTEN, 43-18  
 FPZERO, 43-21  
 FreeFreeList, 29-40  
 FREEHORIZ, 34-631  
 FreeList, 61-88, 61-126  
 FREEVERT, 34-633  
 FreeWObject, 29-40  
 freopen, 55-61  
 frexp, 42-93  
 front, 6-27  
 FrontPen, 34-738, 34-787  
 FRST\_DOT, 27-29, 50-94  
 FS\_NORMAL, 56-17  
 FSB\_BOLD, 56-22  
 FSB\_EXTENDED, 56-18  
 FSB\_ITALIC, 56-20  
 FSB\_UNDERLINED, 56-24  
 fseek, 55-56  
 FSF\_BOLD, 56-23  
 FSF\_EXTENDED, 56-19  
 FSF\_ITALIC, 56-21  
 FSF\_UNDERLINED, 56-25  
 ftell, 55-62  
 function, 3-85  
 GADGBACKFILL, 61-167  
 GADGDISABLED, 34-417  
 Gadget, 33-123, 33-143, 34-220, 34-286, 34-288,  
 34-1165, 34-1358, 34-1490, 34-1573, 61-50,  
 61-51, 61-52, 61-53, 61-54, 61-55, 61-73,  
 61-124  
 GADGET0002, 34-527  
 GADGETCOUNT, 33-57, 33-143  
 GADGETDOWN, 34-979  
 GadgetID, 34-354  
 GADGETON, 33-310  
 GadgetRender, 34-311  
 GadgetReturn, 33-133  
 Gadgets, 34-1573  
 GadgetText, 34-323  
 GadgetType, 34-302, 34-506  
 GADGETUP, 34-981  
 GADGBOX, 34-369  
 GADGCONF, 34-367  
 GADGCHIBITS, 34-366  
 GADGCHIMAGE, 34-371  
 GADGCHNONE, 34-377  
 GADGIMAGE, 34-383

GADGIMMEDIATE, 34-438  
 GamePortTrigger, 23-28  
   GELGONE, 24-30  
   gelHead, 50-42  
   GelsInfo, 50-37, 50-59, 50-59  
   gelTail, 50-42  
   GENLOC, 26-60  
 GENLOCK\_AUDIO, 60-63  
 GENLOCK\_VIDEO, 60-64  
   getc, 55-49, 55-50  
   getchar, 55-50  
   GetIcon, 29-39  
   GetWObject, 29-38  
   GfxBase, 26-24, 33-97, 33-97  
 GID\_DOWNSCROLL, 61-156  
 GID\_HORIZSCROLL, 61-151  
 GID\_LEFTSCROLL, 61-153  
   GID\_NAME, 61-157  
 GID\_RIGHTSCROLL, 61-154  
   GID\_UPSCROLL, 61-155  
 GID\_VERTSCROLL, 61-152  
   GID\_WOBJECT, 61-150  
 GIMMEZEROZERO, 34-1277  
 GOODITEMDRAWN, 33-321  
 GOODSUBDRAWN, 33-322  
 GPCT\_ABSJOYSTICK, 23-41  
 GPCT\_ALLOCATED, 23-36  
   GPCT\_MOUSE, 23-39  
 GPCT\_NOCONTROLLER, 23-37  
 GPCT\_RELJOYSTICK, 23-40  
   GPD\_ASKCTYPE, 23-15  
   GPD\_ASKTRIGGER, 23-17  
   GPD\_READEVENT, 23-14  
   GPD\_SETCTYPE, 23-16  
   GPD\_SETTRIGGER, 23-18  
 GPDERR\_SETCTYPE, 23-45  
   gpt\_Keys, 23-29  
   gpt\_Timeout, 23-30  
   gpt\_XDelta, 23-31  
   gpt\_YDelta, 23-32  
 GPTB\_DOWNKEYS, 23-23  
   GPTB\_UPKEYS, 23-25  
 GPTF\_DOWNKEYS, 23-24  
   GPTF\_UPKEYS, 23-26  
   graphics, 6-5, 16-20, 27-17, 34-25, 34-29, 34-33,  
   34-37, 34-41, 34-45, 35-21, 50-5, 51-5, 60-5  
 GRAPHICS\_CLIP\_H, 6-1, 6-2, 34-28  
 GRAPHICS\_COLLIDE\_H, 8-1, 8-2  
 GRAPHICS\_COPPER\_H, 11-1, 11-2  
 GRAPHICS\_GELS\_H, 24-1, 24-2  
 GRAPHICS\_GFX\_H, 6-4, 25-1, 25-2, 34-24, 50-4, 51-4, 60-4  
 GRAPHICS\_GEXBASE\_H, 26-1, 26-2  
 GRAPHICS\_GFXMACROS\_H, 27-1, 27-2  
 GRAPHICS\_GRAPHINT\_H, 28-1, 28-2  
 GRAPHICS\_LAYERS\_H, 34-40, 39-5, 39-6  
 GRAPHICS\_RASTPORT\_H, 27-16, 34-36, 50-1, 50-2

GRAPHICS\_REGIONS\_H, 51-1, 51-2  
 GRAPHICS\_SPRITE\_H, 53-1, 53-2  
 GRAPHICS\_TEXT\_H, 16-19, 34-44, 56-1, 56-2  
 GRAPHICS\_VIEW\_H, 34-32, 35-20, 60-1, 60-2  
   GRELBOTTOM, 34-393  
   GRELHEIGHT, 34-402  
   GRELRIGHT, 34-395  
   GRELWIDTH, 34-399  
   GRET\_REQSELECT, 33-337  
   GRET\_REQUEST, 33-336  
   GRET\_RJM, 33-335  
   GRET\_RJMSELECT, 33-338  
   GZZGADGET, 34-512  
   GZZHeight, 34-1225  
   GZZMouseX, 34-1218  
   GZZMouseY, 34-1219  
   GZZWidth, 34-1224  
   HAM, 60-60  
 HARDWARE\_ADKBITS\_H, 1-11, 1-12, 1-53  
 HARDWARE\_BLIT\_H, 3-11, 3-12, 3-96  
 HARDWARE\_CUSTOM\_H, 13-11, 13-12, 13-114  
 HARDWARE\_DMABITS\_H, 18-11, 18-12, 18-53  
 HARDWARE\_INTBITS\_H, 32-12, 32-13, 32-53  
   HeadComp, 24-228  
   HeadOb, 24-201  
   Height, 24-104, 33-136, 34-76, 34-120, 34-211,  
   34-293, 34-827, 34-1100, 34-1339, 34-1452,  
   34-1555, 45-85, 53-13  
   HIGHBOX, 34-178  
   HIGHCOMP, 34-176  
   HIGHLAGS, 34-172  
   HIGHIMAGE, 34-174  
   HIGHITEM, 34-189  
   HIGHNONE, 34-180  
   HIRES, 60-58  
   HIRESGADGET, 33-54  
   HIRESPICK, 33-47  
   HitMask, 24-109  
   HitScreen, 33-199  
   HoldMinyMouse, 33-195  
   HOLDNMODIFY, 17-21  
   HorizBody, 34-612  
   HorizPot, 34-570  
   HP\_LASERJET, 34-1802  
   HP\_LASERJET\_PLUS, 34-1803  
   HPotRes, 34-621  
   HSIZEBITS, 3-15  
   HSIZEMASK, 3-17  
   hthick, 33-138  
   HUGE, 42-75  
   HUGE\_VAL, 40-1  
   HWaitPos, 11-35, 11-45, 11-45  
   10, 14-19  
   11, 14-19  
   12, 14-19  
   I\_PI, 42-72



```

I_PID2, 42-73
IAddress, 34-935
IBase-, 33-355, 33-356, 33-357, 33-358, 33-359,
33-360, 33-362, 33-363, 33-364, 33-365,
33-366, 33-367
IBitMap, 33-114
icon, 29-30
ICONNAME, 29-30
id_BytesPerBlock, 19-137
id_DiskState, 19-132
id_DiskType, 19-138
ID_DOS_DISK, 19-157
id_InUse, 19-141
ID_KICKSTART_DISK, 19-164
ID_NO_DISK_PRESENT, 19-155
ID_NOT_REALLY_DOS, 19-158
id_NumBlocks, 19-133
id_NumBlocksUsed, 19-135
id_NumSoftErrors, 19-128
id_UnitNumber, 19-130
ID_UNREADABLE_DISK, 19-156
ID_VALIDATED, 19-151
ID_VALIDATING, 19-149
id_VolumeNode, 19-139
ID_WRITE_PROTECTED, 19-147
IDCMPFlags, 34-1188, 34-1345
IDCMPWindow, 34-956
ie_addr, 31-131, 31-138
ie_Class, 31-122
ie_Code, 31-124
ie_EventAddress, 31-138
ie_NextEvent, 31-121
ie_position, 31-132, 31-136, 31-137, 31-138
ie_Qualifier, 31-125
ie_SubClass, 31-123
ie_TimeStamp, 31-133
ie_x, 31-128, 31-136, 31-136
ie_xy, 31-130, 31-136, 31-137
ie_y, 31-129, 31-137, 31-137
IECLASS_ACTIVEWINDOW, 31-53
IECLASS_CLOSEWINDOW, 31-41
IECLASS_DISKINSERTED, 31-51
IECLASS_DISKREMOVED, 31-49
IECLASS_EVENT, 31-27
IECLASS_GADGETDOWN, 31-33
IECLASS_GADGETUP, 31-35
IECLASS_INACTIVEWINDOW, 31-55
IECLASS_MAX, 10-76, 31-59
IECLASS_MENULIST, 31-39
IECLASS_NEWPREES, 31-47
IECLASS_NULL, 31-21
IECLASS_POINTERPOS, 31-29
IECLASS_RAWKEY, 31-23
IECLASS_RAWMOUSE, 31-25
IECLASS_REFRESHWINDOW, 31-45
IECLASS_REQUESTER, 31-37

```

```

IECLASS_SIZEWINDOW, 31-43
IECLASS_TIMER, 31-31
IECODE_ASCII_DEL, 31-76
IECODE_ASCII_FIRST, 31-74
IECODE_ASCII_LAST, 31-75
IECODE_C0_FIRST, 31-72
IECODE_C0_LAST, 31-73
IECODE_C1_FIRST, 31-77
IECODE_C1_LAST, 31-78
IECODE_COMM_CODE_FIRST, 31-68
IECODE_COMM_CODE_LAST, 31-69
IECODE_KEY_CODE_FIRST, 31-66
IECODE_KEY_CODE_LAST, 31-67
IECODE_LATINI_FIRST, 31-79
IECODE_LATINI_LAST, 31-80
IECODE_LBUTTON, 31-83, 34-1915, 34-1916
IECODE_MBUTTON, 31-85
IECODE_NEWACTIVE, 31-89
IECODE_NOBUTTON, 31-86
IECODE_RBUTTON, 31-84, 34-1917, 34-1918
IECODE_REOCLEAR, 31-97
IECODE_REQSET, 31-95
IECODE_UP_PREFIX, 31-65, 34-1915, 34-1917
IEQUALIFIER_CAPSLOCK, 31-103, 33-327
IEQUALIFIER_CONTROL, 31-104
IEQUALIFIER_INTERRUPT, 31-111
IEQUALIFIER_LALT, 31-105, 34-1923
IEQUALIFIER_LBUTTON, 31-113
IEQUALIFIER_LCOMMAND, 31-107, 34-1925
IEQUALIFIER_LSHIFT, 31-101, 33-327
IEQUALIFIER_MBUTTON, 31-115
IEQUALIFIER_MULTIBROADCAST, 31-112
IEQUALIFIER_NUMERICPAD, 31-109
IEQUALIFIER_RALT, 31-106, 34-1924
IEQUALIFIER_RBUTTON, 31-114
IEQUALIFIER_RCOMMAND, 31-108, 34-1926
IEQUALIFIER_RELATIVEMOUSE, 31-116
IEQUALIFIER_REPEAT, 31-110
IEQUALIFIER_RSHIFT, 31-102, 33-327
if, 36-31, 55-36
Image, 33-125, 33-144, 34-819, 34-895, 34-1203,
34-1367, 61-56, 61-57
IMAGE_NEGATIVE, 34-1771
IMAGE_POSITIVE, 34-1770
ImageBMap, 34-245
ImageData, 24-111, 34-828
ImageShadow, 24-154
INACTIVEWINDOW, 34-1007
IND_ADDHANDLER, 30-16
IND_REMHANDLER, 30-17
IND_SETMPORT, 30-21
IND_SETMTRIC, 30-23
IND_SETMTYPE, 30-22
IND_SETPERIOD, 30-20
IND_SETTHRESH, 30-19
IND_WRITEEVENT, 30-18

```

```

InfoData, 19-127
INGADGETSTATE, 33-317
InitAnimate, 24-253
INMENUSTATE, 33-318
innerHeight, 33-140
innerWidth, 33-140
inputevent, 10-24, 31-120, 31-121, 33-121, 34-61
InputInterrupt, 33-118
InputRequest, 33-117
INREQUEST, 34-1293
INTB_AUD0, 32-25
INTB_AUD1, 32-24
INTB_AUD2, 32-23
INTB_AUD3, 32-22
INTB_BLIT, 32-26
INTB_COPER, 32-28
INTB_DSKBLK, 32-31
INTB_DSKSYNC, 32-20
INTB_EXTER, 32-19
INTB_INTEN, 32-18
INTB_PORTS, 32-29
INTB_RBF, 32-21
INTB_SETCLR, 32-15
INTB_SOFTINT, 32-30
INTB_TBE, 32-32
INTB_VERTB, 32-27
intena, 13-83, 27-25, 27-26
intena, 13-35
INTERLACE, 17-22
Interrupt, 15-51, 15-52, 15-53, 15-65, 15-66, 15-67,
26-35, 33-118
INTE_AUD0, 32-44
INTE_AUD1, 32-43
INTE_AUD2, 32-42
INTE_AUD3, 32-41
INTE_BLIT, 32-45
INTE_COPER, 32-47
INTE_DSKBLK, 32-50
INTE_DSKSYNC, 32-39
INTE_EXTER, 32-38
INTE_INTEN, 32-37
INTE_PORTS, 32-48
INTE_RBF, 32-40
INTE_SETCLR, 32-36
INTE_SOFTINT, 32-49
INTE_TBE, 32-51
INTE_VERTB, 27-25, 27-26, 32-46
intreq, 13-84
intreq, 13-36
IntuEvents, 33-121
IntuiMessage, 33-198, 34-913, 34-959, 34-1191
IntuiText, 34-224, 34-323, 34-736, 34-754
INTUITICKS, 34-1012
INTUITION_INTUINTERNAL_H, 33-2, 33-3
INTUITION_INTUITION_H, 34-4, 34-5, 49-43, 61-30, 61-32
INTUITION_INTUITIONBASE_H, 33-18, 34-20, 35-1, 35-2

```

```

IntuitionBase, 33-67, 34-21, 35-38
INVERSVID, 50-91
io_Actual, 7-44
io_Baud, 52-65
io_BrkTime, 52-66
io_ClipID, 7-48
io_ColorMap, 48-145
io_Command, 7-41, 48-127, 48-141
io_CtlChar, 52-62
io_Data, 7-46
io_DestCols, 48-151
io_DestRows, 48-152
io_Device, 7-39, 48-125, 48-139
io_Error, 7-43, 48-129, 48-143
io_ExtFlags, 52-64
io_Flags, 7-42, 48-128, 48-142
io_Length, 7-45
io_Message, 7-38, 48-124, 48-138
io_Modes, 48-146
io_Offset, 7-47
io_ParFlags, 46-60
io_Parm0, 48-131
io_Parm1, 48-132
io_Parm2, 48-133
io_Parm3, 48-134
io_PExtFlags, 46-58
io_PrtCommand, 48-130
io_PTermArray, 46-61
io_RastPort, 48-144
io_RBufLen, 52-63
io_ReadLen, 52-68
io_SerFlags, 52-71
io_Special, 48-153
io_SrcHeight, 48-150
io_SrcWidth, 48-149
io_SrcX, 48-147
io_SrcY, 48-148
io_Status, 46-59, 52-72
io_StopBits, 52-70
io_TermArray, 52-67
io_Unit, 7-40, 48-126, 48-140
io_WriteLen, 52-69
ioa_AllocKey, 2-44
ioa_Cycles, 2-49
ioa_Data, 2-45
ioa_Length, 2-46
ioa_Period, 2-47
ioa_Request, 2-43
ioa_Volume, 2-48
ioa_WriteMag, 2-50
IOAudio, 2-42
IOClipReq, 7-37
IODRPRReq, 48-137
IOExcess, 33-193
IOExtPar, 46-35, 49-70, 49-76
IOExtSer, 49-71, 49-77, 52-35

```

IOExtID, 58-109  
 IOPar, 46-36  
 IOPARB\_ABORT, 46-72  
 IOPARB\_ACTIVE, 46-74  
 IOPARB\_QUEUED, 46-70  
 IOPARB\_ABORT, 46-73  
 IOPARB\_ACTIVE, 46-75  
 IOPARB\_QUEUED, 46-71  
 IOArray, 46-25, 46-61  
 IOprtCmdReq, 48-123  
 IOPTB\_PAPEROUT, 46-80  
 IOPTB\_PBUSY, 46-78  
 IOPTB\_PSEL, 46-82  
 IOPTB\_RWDIR, 46-76  
 IOPTF\_PAPEROUT, 46-81  
 IOPTF\_PBUSY, 46-79  
 IOPTF\_PSEL, 46-83  
 IOPTF\_RWDIR, 46-77  
 IOrequest, 2-43, 57-34  
 IOSERB\_ABORT, 52-116  
 IOSERB\_ACTIVE, 52-118  
 IOSERB\_BUFFERREAD, 52-112  
 IOSERB\_QUEUED, 52-114  
 IOSERF\_ABORT, 52-117  
 IOSERF\_ACTIVE, 52-119  
 IOSERF\_BUFFERREAD, 52-113  
 IOSERF\_QUEUED, 52-115  
 IOSTB\_OVERRUN, 52-128  
 IOSTB\_READBREAK, 52-124  
 IOSTB\_WRITEBREAK, 52-126  
 IOSTB\_XOFFREAD, 52-120  
 IOSTB\_XOFFWRITE, 52-122  
 IOSTdReq, 33-117, 45-63, 46-36, 52-36, 58-110  
 IOSTF\_OVERRUN, 52-129  
 IOSTF\_READBREAK, 52-125  
 IOSTF\_WRITEBREAK, 52-127  
 IOSTF\_XOFFREAD, 52-121  
 IOSTF\_XOFFWRITE, 52-123  
 IOTArray, 52-26, 52-67  
 iotd\_Count, 58-111  
 iotd\_Req, 58-110  
 iotd\_SecLabel, 58-112  
 IPointer, 33-149  
 IPOINHEIGHT, 33-416  
 IPOINTHOTX, 33-417  
 IPOINTHOTY, 33-418  
 Iptr, 28-16  
 IPtrHeight, 33-150  
 IPtrWidth, 33-151  
 iqd\_ENKUHDPort, 33-197  
 is\_Node, 28-15  
 isalnum, 12-42, 12-47  
 isalpha, 12-35, 12-48  
 isascii, 12-46  
 iscntrl, 12-45  
 iscsym, 12-47

iscsymf, 12-48  
 isdigit, 12-38  
 ISDRAWN, 34-187  
 isgraph, 12-44  
 ISCRTRX, 6-79  
 ISCRTRY, 6-80  
 ISLESSX, 6-77  
 ISLESSY, 6-78  
 islower, 12-37, 12-50  
 isprint, 12-43  
 ispunct, 12-41  
 Isrvstr, 28-13, 28-16  
 isspace, 12-40  
 isupper, 12-36, 12-51  
 isxdigit, 12-39  
 ItemCRect, 33-112  
 ITEMDRAWN, 33-315  
 ITEMENABLED, 34-167  
 ItemFill, 34-129  
 ITEMNUM, 34-1846  
 ITEMTEXT, 34-157  
 IText, 34-752  
 ITextFont, 34-750  
 itof, 43-25  
 IXOffset, 33-152  
 IYOffset, 33-152  
 JAM1, 50-88  
 JAM2, 34-1906, 50-89  
 JazzX, 34-87  
 JazzY, 34-87  
 joy0dat, 13-26  
 joy1dat, 13-27  
 joytest, 13-47  
 jrand48, 42-91  
 K, 4-28, 4-28, 4-31, 4-31, 19-164, 19-165  
 KARLA, 33-422  
 KBD\_ADDRESETHANDLER, 37-19  
 KBD\_READEVENT, 37-17  
 KBD\_READMATRIX, 37-18  
 KBD\_REMRESETHANDLER, 37-20  
 KBD\_RESETHANDLERDONE, 37-21  
 KC\_NOQUAL, 38-27  
 KC\_VANILLA, 38-28  
 KCB\_CONTROL, 38-31  
 KCB\_DOWNUP, 38-33  
 KCB\_NOP, 38-24  
 KCB\_STRING, 38-36  
 KCF\_ALT, 38-30  
 KCF\_CONTROL, 38-32  
 KCF\_DOWNUP, 38-34  
 KCF\_NOP, 38-25  
 KCF\_SHIFT, 38-29  
 KCF\_STRING, 38-37  
 KEYCODE\_M, 34-1936  
 KEYCODE\_N, 34-1935  
 KEYCODE\_Q, 34-1933

KEYCODE\_X, 34-1934  
 KEYDELMIC, 33-252  
 KEYDELSEC, 33-251  
 Keymap, 10-20, 10-53, 33-187, 33-187, 34-716, 38-13  
 KEYREPMIC, 33-250  
 KEYREPSEC, 33-249  
 KeyRptDelay, 34-1649  
 KeyRptSpeed, 34-1647  
 km\_HiCapsable, 38-20  
 km\_HiKeyMap, 38-19  
 km\_HiKeyMapTypes, 38-18  
 km\_HiRepeatable, 38-21  
 km\_LoCapsable, 38-16  
 km\_LoKeyMap, 38-15  
 km\_LoKeyMapTypes, 38-14  
 km\_LoRepeatable, 38-17  
 KNOBHIT, 34-637  
 KNOBHMIN, 34-644  
 KNOBVMIN, 34-646  
 l\_LockMessage, 6-48  
 LACE, 60-59  
 lastBlissObj, 50-49  
 lastColor, 50-46  
 Layer, 6-25, 6-27, 6-63, 33-135, 33-237, 33-237,  
 34-231, 34-691, 34-1241, 34-1506, 39-24,  
 39-25, 39-26, 50-54, 50-54  
 Layer\_Info, 6-52, 34-1482, 39-22  
 LAYERBACKDROP, 39-19  
 LayerInfo, 6-52, 34-1482  
 LayerInfo\_extra, 39-39, 39-39  
 LayerInfo\_extra\_size, 39-37  
 LayerLockCount, 6-35  
 LayerLocker, 6-53  
 LayerPtr, 34-691  
 LAYERREFRESH, 39-20  
 layers, 34-41  
 LAYERSIMPLE, 39-16  
 LAYERSMART, 39-17  
 LAYERSUPER, 39-18  
 ldexp, 42-16, 42-93  
 left, 33-136  
 LEFTBORDER, 34-474, 34-622  
 LeftEdge, 34-74, 34-118, 34-209, 34-291, 34-742,  
 34-785, 34-821, 34-1098, 34-1337, 34-1450,  
 34-1555  
 LEFTHIT, 8-35  
 leftmost, 50-48  
 LETTER, 34-1763  
 LIB\_VECSIZE, 44-48  
 LIB\_VECTSIZE, 15-110, 15-111, 15-112, 15-113, 15-114  
 LibNode, 26-26, 33-72, 35-40  
 LIBRARIES\_DISKFONT\_H, 16-1, 16-2  
 LIBRARIES\_DOS\_H, 19-4, 19-5, 19-229, 20-23, 54-17, 54-19  
 LIBRARIES\_DOSEXTENS\_H, 20-2, 20-3, 20-233  
 LIBRARIES\_ICON\_H, 29-2, 29-3, 29-44  
 LIBRARIES\_MATHEFP\_H, 43-1, 43-2, 43-46

LIBRARIES\_TRANSLATOR\_H, 59-1, 59-2, 59-15  
 LINEMODE, 3-59  
 LinePtrn, 27-29, 50-69  
 linpatcnt, 50-66  
 List, 15-64, 26-36, 26-51, 26-53, 61-62, 61-90  
 lobs, 6-63  
 Lock, 6-31, 39-29  
 LockCount, 6-33  
 Locker, 39-33  
 LockMessage, 6-46  
 LockNest, 39-31  
 LockPort, 6-45, 39-28  
 LOFCprList, 60-48  
 LOFList, 26-31  
 log, 42-17, 42-92  
 log10, 42-18, 42-92, 43-16  
 LOGHUGE, 42-77  
 LOGTINY, 42-78  
 LONELYMESSAGE, 34-1063  
 LONGINT, 34-490, 34-706  
 longreserved, 39-38, 50-83  
 LOWCHECKWIDTH, 34-1879  
 LOWCOMMWIDTH, 34-1880  
 LOWRESGADGET, 33-55  
 LOWRESPICK, 33-48  
 lrand48, 42-91  
 M, 14-23, 27-30, 27-30  
 M\_ASM, 9-93  
 M\_AWM, 9-94  
 M\_LNM, 9-92, 10-27  
 MALE, 45-38, 45-42  
 MAlloc, 61-134  
 Mask, 27-30, 33-236, 50-60  
 MatchToolValue, 29-39  
 matherr, 42-90  
 max, 11-54, 41-4, 55-66  
 MAXBODY, 34-648  
 MAXBYTESPERROW, 3-20  
 MaxChars, 34-676  
 MaxCount, 11-67, 50-25  
 MAXCMLS, 58-35  
 MAXDISPLAYCOLUMNS, 33-406  
 MAXDISPLAYHEIGHT, 33-403, 33-404  
 MAXDISPLAYROWS, 33-404  
 MAXDISPLAYWIDTH, 33-405, 33-406  
 MAXFONTNAME, 16-41, 16-55  
 MAXFONTPATH, 16-23, 16-26  
 MAXREQ, 45-54  
 MaxHeight, 34-1107, 34-1415  
 MAXINT, 19-47  
 MAXPITCH, 45-52  
 MAXPOT, 34-649  
 MAXRATE, 45-50  
 MAXRETRY, 58-38  
 MAXTABS, 10-29, 10-55  
 MAXVOL, 45-56

```

MaxWidth, 34-1107, 34-1415
MaxX, 25-28
MaxXMouse, 33-92
MaxY, 25-28
MaxYMouse, 33-93
MeMask, 24-108
Memory, 34-1829
Menu, 34-71, 34-73, 34-1112
MENUCANCEL, 34-1075
MENUDOWN, 34-1918
MenuDrawn, 33-94, 33-355, 33-356, 33-357, 33-358,
33-359, 33-360
MENUENABLED, 34-92
MenuHBorder, 33-164, 34-1469
MENUHOT, 34-1073
MenuItem, 34-82, 34-114, 34-116, 34-143
MenuName, 34-80
MENUNULL, 34-1858
MENUNUM, 34-1845
MENUPICK, 34-985
MenuRPort, 33-111
MenuSelected, 33-95, 33-362, 33-363, 33-364, 33-365,
33-366, 33-367
MENUSTATE, 34-1295
MenuStrip, 34-1112
MENUTOGGLE, 34-165
MENUTOGGLED, 34-191
MENUUP, 34-1917
MenuVBorder, 33-163, 34-1468
MENUVERIFY, 34-995
MENUWAITING, 34-1077
Message, 2-50, 6-46, 6-48, 7-38, 7-54, 15-50, 20-59,
20-78, 20-109, 34-915, 45-63, 48-124, 48-138,
54-22, 56-56
MessageKey, 34-1191
Micros, 33-90, 34-950
MIDRAWN, 34-96
MIN, 41-5, 55-67
MINEREQ, 45-53
MinHeight, 34-1106, 34-1414
MININT, 19-48
MINPITCH, 45-51
MINRATE, 45-49
minters, 50-71
MINVOL, 45-55
MinWidth, 34-1106, 34-1414
MinX, 25-27
MinXMouse, 33-92
MinY, 25-27
MinYMouse, 33-93
misc, 44-51
MISCNAME, 44-51
MiscResource, 44-42
Mode, 33-236, 45-66
MODE_640, 17-14
MODE_NEWFILE, 19-28

```

```

MODE_OLDFILE, 19-24
Modes, 26-38, 60-40, 60-52
modf, 42-19, 42-93
MOUSEBUTTONS, 34-975
MOUSEDBLMIC, 33-254
MOUSEDBLSEC, 33-253
MOUSEMOVE, 34-977
MouseX, 33-87, 34-944, 34-1103, 34-1455
MouseY, 33-87, 34-944, 34-1103, 34-1455
MouseYMinimum, 33-189
mouth_rb, 45-82
mouths, 45-72
mr_AllocArray, 44-44
MR_ALLOCMISCRESOURCE, 44-47
MR_FREEMISCRESOURCE, 44-48
mr_Library, 44-43
MR_PARALLELBITS, 44-38
MR_PARALLELPORT, 44-37
MR_SERIALBITS, 44-36
MR_SERIALPORT, 44-35
mrans48, 42-91
MSDOS1, 36-31
MsgPort, 6-45, 6-47, 10-33, 20-33, 20-60, 20-61,
20-79, 20-209, 20-229, 33-197, 34-1190,
39-27, 39-28, 49-62, 49-82, 54-23
MTYPE_CLOSEDOWN, 61-146
MTYPE_DISKCHANCE, 61-144
MTYPE_IOPROC, 61-147
MTYPE_PSTD, 61-142
MTYPE_TIMER, 61-145
MTYPE_TOOLEXIT, 61-143
MUSTDRAW, 24-26
MutualExclude, 34-125, 34-344
N_TRACTOR, 33-292, 34-1785
NABC, 3-28, 3-34, 3-35, 3-36
NABNC, 3-29, 3-34
NANBC, 3-30, 3-35, 3-36
NANBNC, 3-31
narrator_rb, 45-62, 45-83
NATURALE0, 45-40, 45-43
ND_CantAlloc, 45-18
ND_Expunged, 45-21
ND_FreqErr, 45-27
ND_MakeBad, 45-16
ND_ModeErr, 45-26
ND_NoAudLib, 45-15
ND_NoMem, 45-14
ND_NoWrite, 45-20
ND_PhonErr, 45-22
ND_PitchErr, 45-24
ND_RateErr, 45-23
ND_SexErr, 45-25
ND_Unimpl, 45-19
ND_UnitErr, 45-17
ND_VolErr, 45-28
NEWCLIPRECTS_1_1, 6-68

```

```

newlayer, 33-135
NEWLAYERINFO_CALLED, 39-42
NEWPREES, 34-997
NewScreen, 34-1553
NEWSIZE, 34-971
NewWindow, 34-1331, 61-43, 61-67
Next, 6-61, 11-52, 11-59, 11-73, 51-14, 60-31, 60-69
NextBorder, 34-796
NextComp, 24-189
NextGadget, 34-288
NextImage, 34-895
NextItem, 34-116
nextLine, 50-44
NextMenu, 34-73
NextOb, 24-209
NextRemember, 34-1827
NextScreen, 34-1445
NextSelect, 34-150
NextSeq, 24-193
NextText, 34-754
NextVSprite, 24-79
NextWindow, 34-1095
nm_masks, 45-69
NO_ICON_POSITION, 61-172
NOCAREREFRESH, 34-1302
NOCONSOLE, 33-393
NOCROSSFILL, 50-102
Node, 7-31, 16-51, 28-15, 61-94, 61-95, 61-96, 61-97
nodes, 7-14, 16-14, 28-9, 48-20, 49-16, 61-19
NOGRAPHICS, 33-386
NOITEM, 34-1856
NOLAYERS, 33-387
NOMENU, 34-1855
NONDP, 42-7
NOSUB, 34-1857
NOT, 34-1865
nrand48, 42-91
NTSC, 26-59
NUEBS, 36-12
NULL, 24-253, 34-1909, 34-1910, 55-35, 55-37, 55-39
num, 53-15
numchan, 45-74
NumChars, 34-685
NUMCYLS, 58-34, 58-35, 58-39
NUMHEADS, 58-37, 58-39
NUMRTYPES, 44-40, 44-44
NUMSECS, 58-36
NUMTRACKS, 58-39
NUMUNITS, 58-40
nxtlist, 11-25, 11-42, 11-42
O, 4-27, 4-30, 19-157, 19-158
O_APPEND, 22-11
O_CREAT, 22-12
O_EXCL, 22-14
O_NDELAY, 22-10
O_RAW, 22-16

```

```

O_RDONLY, 22-6
O_RDWR, 22-8
O_TRUNC, 22-13
O_WRONLY, 22-7
OAlloc, 61-135
obj, 61-135, 61-135
ObjAlloc, 61-135
obs, 39-26
OCTANT1, 3-79
OCTANT2, 3-78
OCTANT3, 3-77
OCTANT4, 3-76
OCTANT5, 3-75
OCTANT6, 3-74
OCTANT7, 3-73
OCTANT8, 3-72
OFF_DISPLAY, 27-21
OFF_SPRITE, 27-23
OFF_VBLANK, 27-26
OFFSET_BEGINING, 19-41
OFFSET_BEGINNING, 19-34, 19-41
OFFSET_CURRENT, 19-36
OFFSET_END, 19-38
OKIMATE_20, 34-1799
OlderRequest, 34-208
OldX, 24-92, 33-137
OldY, 24-92, 33-137
ON_DISPLAY, 27-20
ON_SPRITE, 27-22
ON_VBLANK, 27-25
ONE_DOT, 50-95
ONEDOT, 3-63
OpCode, 11-22
OptionList, 33-131
OTHER_REFRESH, 34-1269
OUTSTEP, 24-44
OVERFLOW, 42-49
OVERLAY, 24-25
OVERLAG, 3-64
P_STKSIZE, 49-58, 49-84
pad, 25-39, 45-75, 45-87
pad2d, 13-61
pad34, 13-66
pad3b, 13-70
pad7c, 13-95
pad83, 13-99
pad86, 13-102
pad8e, 13-104
padding, 34-1729
PAL, 26-61
PAPERLENGTH, 33-294, 34-1724
PAPERSIZE, 33-292, 34-1722
PAPERTYPE, 33-295, 34-1726
PAPERWIDTH, 33-293
parallel, 46-85, 49-32
PARALLEL_PRINTER, 34-1735

```

PARALLELNAME, 46-85  
 PARB\_EOFMODE, 46-68  
 PARB\_RAD\_BOOGIE, 46-66  
 PARB\_SHARED, 46-64  
 Parent, 34-1168  
 ParErr\_BufTooBig, 46-91  
 ParErr\_DevBusy, 46-90  
 ParErr\_InitErr, 46-96  
 ParErr\_InvParam, 46-92  
 ParErr\_LineErr, 46-93  
 ParErr\_NotOpen, 46-94  
 ParErr\_PortReset, 46-95  
 PARE\_EOFMODE, 46-69  
 PARE\_RAD\_BOOGIE, 46-67  
 PARE\_SHARED, 46-65  
 PCC\_BW, 49-100  
 PCC\_YMC, 49-101  
 PCC\_YMC\_BW, 49-102  
 PCC\_YMCB, 49-103  
 pd\_Device, 49-61  
 pd\_Flags, 49-85  
 pd\_ior0, 49-72, 49-73, 49-74  
 pd\_ior1, 49-78, 49-79, 49-80  
 pd\_IORPort, 49-82  
 pd\_p0, 49-70, 49-73  
 pd\_p1, 49-76, 49-79  
 pd\_pad, 49-86  
 pd\_PBothReady, 49-68  
 pd\_PIOR0, 49-73  
 pd\_PIOR1, 49-79  
 pd\_Preferences, 49-87  
 pd\_PrintBuf, 49-66  
 pd\_PrinterSegment, 49-63  
 pd\_PrinterType, 49-64  
 pd\_PWaitEnabled, 49-88  
 pd\_PWrite, 49-67  
 pd\_s0, 49-71, 49-74  
 pd\_s1, 49-77, 49-80  
 pd\_SegmentData, 49-65  
 pd\_SIOR0, 49-74  
 pd\_SIOR1, 49-80  
 pd\_Stk, 49-84  
 pd\_TC, 49-83  
 pd\_TIOR, 49-81  
 pd\_Unit, 49-82  
 PDCMD\_QUERY, 46-87  
 PDCMD\_SETPARAMS, 46-88  
 PDERR\_BADDIMENSION, 48-172  
 PDERR\_BUFFERMEMORY, 48-175  
 PDERR\_CANCEL, 48-169  
 PDERR\_DIMENSIONOVLW, 48-173  
 PDERR\_INTERNALMEMORY, 48-174  
 PDERR\_INVERTHAM, 48-171  
 PDERR\_NOTGRAPHICS, 48-170  
 ped\_Close, 49-110  
 ped\_ColorClass, 49-112

ped\_Commands, 49-120  
 ped\_DoSpecial, 49-121  
 ped\_Expunge, 49-108  
 ped\_Init, 49-107  
 ped\_MaxColumns, 49-113  
 ped\_MaxXDots, 49-116  
 ped\_MaxYDots, 49-117  
 ped\_NumCharSets, 49-114  
 ped\_NumRovs, 49-115  
 ped\_Open, 49-109  
 ped\_PrinterClass, 49-111  
 ped\_PrinterName, 49-106  
 ped\_Render, 49-122  
 ped\_TimeoutSecs, 49-123  
 ped\_XDotsInch, 49-118  
 ped\_YDotsInch, 49-119  
 PenHeight, 50-73  
 PenWidth, 50-72  
 PF2PRI, 17-18  
 PF\_FINE\_SCROLL\_MASK, 17-27  
 PFA\_FINE\_SCROLL, 17-25  
 PFB\_FINE\_SCROLL\_SHIFT, 17-26  
 PFBA, 60-56  
 PI, 14-21, 42-69, 43-11, 43-12, 43-13, 43-14  
 PI2, 43-13  
 PI4, 43-14  
 PICA, 33-281, 34-1753  
 PID2, 14-21, 42-70  
 PID4, 42-71  
 PIM2, 14-21  
 pitch, 45-65  
 PlaneOnOff, 24-137, 34-887  
 PlanePick, 24-136, 34-887  
 PLANEPTR, 25-31, 25-40  
 Planes, 25-40  
 PLNCNTMSK, 17-15  
 PLNCNTSHET, 17-17  
 PLOSS, 42-52  
 PMB\_ASM, 10-27, 10-28  
 PMB\_AWM, 10-28, 10-75  
 Pointer, 34-1177  
 PointerMatrix, 34-1659  
 POINTERMATRIXMINREQ, 33-36  
 POINTERSIZE, 34-1608, 34-1659  
 POINTERTICKS, 33-262, 34-1669  
 POINTERX, 33-257  
 POINTERY, 33-258  
 POINTREL, 34-254  
 pos, 13-107  
 posctldata, 53-12  
 pot0dat, 13-30  
 pot1dat, 13-31  
 potgo, 13-46, 47-7  
 POTGONAME, 47-7  
 potinp, 13-32  
 pow, 42-20, 42-92

pow2, 42-21  
 PPC\_BWALPHA, 49-96  
 PPC\_BWGEX, 49-97  
 PPC\_COLORGEX, 49-98  
 PPCB\_COLOR, 49-93  
 PPCB\_GEX, 49-91  
 PPCE\_COLOR, 49-94  
 PPCE\_GEX, 49-92  
 pr\_CIS, 20-42  
 pr\_CLI, 20-47  
 pr\_ConsoleTask, 20-44  
 pr\_COS, 20-43  
 pr\_CurrentDir, 20-41  
 pr\_FileSystemTask, 20-46  
 pr\_GlobVec, 20-37  
 pr\_MsgPort, 20-33  
 pr\_Pad, 20-34  
 pr\_PktWait, 20-49  
 pr\_Result2, 20-40  
 pr\_ReturnAddr, 20-48  
 pr\_SegList, 20-35  
 pr\_StackBase, 20-39  
 pr\_StackSize, 20-36  
 pr\_Task, 20-32  
 pr\_TaskNum, 20-38  
 pr\_WindowPtr, 20-50  
 PRD\_DUMERPORT, 48-33  
 PRD\_PRTCOMMAND, 48-32  
 PRD\_RAWWRITE, 48-31  
 PREDRAWN, 34-256  
 PREE\_FILE, 33-298  
 Preferences, 33-179, 33-179, 34-1631, 49-87  
 prefs, 33-298  
 prev, 6-62, 51-14  
 PrevComp, 24-190  
 PrevItem, 33-129  
 PrevObj, 24-209  
 PrevSeq, 24-194  
 PrevVSprite, 24-80  
 PRIMARY\_CLIP, 7-51  
 PRINTASPECT, 33-287, 34-1714  
 PrinterData, 49-60  
 PrinterExtendedData, 49-105, 49-131  
 PrinterFilename, 34-1694  
 PrinterPort, 34-1639  
 PrinterSegment, 49-65, 49-126  
 PRINTERTYPE, 33-278, 34-1692  
 PrintImage, 34-1712  
 PRINTINVERSE, 33-286  
 PRINTLEFTMARGIN, 33-284, 34-1704  
 PRINTPITCH, 33-281, 34-1698  
 PRINTQUALITY, 33-282, 34-1700  
 PRINTRIGHTMARGIN, 33-285, 34-1710  
 PRINTSHADE, 33-288, 34-1716  
 PRINTSPACING, 33-283, 34-1702  
 PRINTTHRESHOLD, 33-289, 34-1718

Process, 20-31  
 PROPBORDERLESS, 34-635  
 PROPGADGET, 34-528  
 PropInfo, 33-124, 34-551, 61-58, 61-59  
 ps\_NextSegment, 49-127  
 ps\_PED, 49-131  
 ps\_Revision, 49-130  
 ps\_runAlert, 49-128  
 ps\_Version, 49-129  
 PTermArray0, 46-26  
 PTermArray1, 46-27  
 PtrHeight, 34-1178  
 PtrWidth, 34-1181  
 putc, 55-51, 55-52  
 putchar, 55-52  
 PutIcon, 29-39  
 PutWObject, 29-39  
 Qualifier, 34-929  
 QUME\_LP\_20, 34-1800  
 R, 20-123  
 RasInfo, 60-42, 60-42, 60-67, 60-69  
 RasPtr, 50-31  
 RASSIZE, 25-43  
 RastPort, 6-29, 27-17, 33-111, 33-134, 34-37, 34-1133,  
 34-1155, 34-1478, 34-1478, 48-144, 50-52  
 rate, 45-64  
 RAWKEY, 34-989  
 RECOVERY\_ALERT, 34-1890  
 Rectangle, 6-30, 6-65, 25-25, 51-15, 51-20  
 refptr, 13-40  
 REFRESHBITS, 34-1265  
 REFRESHWINDOW, 34-973  
 Region, 6-49, 51-18  
 RegionRectangle, 51-12, 51-14, 51-21, 51-21  
 RELEASED, 33-319  
 RelLeft, 34-217  
 RelTop, 34-217  
 RELVERIFY, 34-430  
 RemBob, 24-254  
 Remember, 34-1825, 34-1827  
 RememberSize, 34-1828  
 ReplyPort, 6-47  
 REPORTMOUSE, 34-1274  
 REQACTIVE, 34-264  
 ReqBorder, 34-222  
 REQCLEAR, 34-993  
 ReqCount, 34-1128  
 ReqGadget, 34-220, 34-514  
 ReqLayer, 34-231  
 REQOFFWINDOW, 34-262  
 ReqPad1, 34-233  
 ReqPad2, 34-249  
 REQSET, 34-983  
 ReqText, 34-224  
 REQUESTDEST, 33-375  
 Requester, 34-204, 34-208, 34-1118, 34-1125



RECOVERIFY, 34-991  
 RESCOUNT, 33-53, 33-143, 33-144, 33-144  
 reserved, 6-36, 6-67, 26-56, 50-84, 60-41  
 reserved1, 6-37  
 resource, 5-11, 5-12, 15-107, 44-51, 47-7  
 RESOURCES\_DISK\_H, 15-2, 15-3, 15-129  
 RESOURCES\_MISC\_H, 44-53  
 RESOURCES\_MISC\_I, 44-1, 44-2  
 RESOURCES\_POTGO\_H, 47-1, 47-2  
 RESTORING, 33-240  
 RETURN\_ERROR, 19-204  
 RETURN\_FAIL, 19-206  
 RETURN\_OK, 19-200  
 RETURN\_WARN, 19-202  
 retval, 42-39  
 rewind, 55-56  
 RIGHTBORDER, 34-473  
 RIGHTHIT, 8-36  
 rightmost, 50-48  
 RINGTRIGGER, 24-49  
 RingXTrans, 24-223  
 RingYTrans, 24-223  
 RMBTRAP, 34-1300  
 rn\_ConsoleSegment, 20-164  
 rn\_Info, 20-167  
 rn\_RestartSeg, 20-166  
 rn\_TaskArray, 20-161  
 rn\_Time, 20-165  
 ROBOTICF0, 45-41  
 RootNode, 20-160  
 round, 43-24  
 Rows, 25-36  
 rp, 6-29, 33-134  
 RP\_ReplyPort, 39-27  
 RP\_User, 50-81  
 RPD, 14-24  
 RPort, 34-1133  
 RWindow, 34-247  
 RxOffset, 60-71  
 RyOffset, 60-71  
 sampfreq, 45-71  
 SatisfyMsg, 7-53  
 SAVEBACK, 24-24  
 SAVEBOB, 24-36  
 SaveBuffer, 24-151  
 SaveColor0, 34-1503  
 savelayer, 33-135  
 SAVEPRESERVE, 24-43  
 SAVERMOUSE, 33-311  
 SaveRPort, 33-234  
 SAVING, 33-239  
 SBitMap, 33-115  
 Screen, 33-79, 33-84, 33-132, 33-199, 33-210,  
 34-1131, 34-1380, 34-1380, 34-1443, 34-1445,  
 35-45, 35-50  
 SCREENDEST, 33-373

ScreenTitle, 34-1205  
 SCREENTYPE, 34-1521  
 SCRGADGET, 34-510  
 Scroll\_X, 6-44  
 Scroll\_Y, 6-44  
 SDCMD\_BREAK, 52-93  
 SDCMD\_QUERY, 52-92  
 SDCMD\_SETPARAMS, 52-94  
 SDOWNBACK, 34-523  
 SDOWNBACKGADGET, 33-64  
 SDRAGGADGET, 33-65  
 SDRAGGING, 34-519  
 Seconds, 33-89, 34-950  
 seed48, 42-89  
 SELECTDOWN, 34-1916  
 Selected, 33-128, 34-408  
 SelectFill, 34-137  
 SelectRender, 34-317  
 SELECTUP, 34-1915  
 SERB\_7WIRE, 52-106  
 SERB\_EOEMODE, 52-98  
 SERB\_PARTY\_ODD, 52-108  
 SERB\_PARTY\_ON, 52-110  
 SERB\_QUEUEDRK, 52-104  
 SERB\_RAD\_BOOGIE, 52-102  
 SERB\_SHARED, 52-100  
 SERB\_XDISABLED, 52-96  
 serdat, 13-44  
 serdatr, 13-33  
 SerErr\_BaudMismatch, 52-132  
 SerErr\_BuFErr, 52-134  
 SerErr\_BufOverflow, 52-142  
 SerErr\_DetectedBreak, 52-145  
 SerErr\_DevBusy, 52-131  
 SerErr\_InitErr, 52-140  
 SerErr\_InvBaud, 52-133  
 SerErr\_InvParam, 52-135  
 SerErr\_LineErr, 52-136  
 SerErr\_NCTS, 52-144  
 SerErr\_NODSR, 52-143  
 SerErr\_NotOpen, 52-137  
 SerErr\_ParityErr, 52-139  
 SerErr\_PortReset, 52-138  
 SerErr\_TimerErr, 52-141  
 SERE\_7WIRE, 52-107  
 SERE\_EOEMODE, 52-99  
 SERE\_PARTY\_ODD, 52-109  
 SERE\_PARTY\_ON, 52-111  
 SERE\_QUEUEDRK, 52-105  
 SERE\_RAD\_BOOGIE, 52-103  
 SERE\_SHARED, 52-101  
 SERE\_XDISABLED, 52-97  
 serial, 49-35, 52-147  
 SERIAL\_PRINTER, 34-1736  
 SERIALNAME, 52-147  
 serper, 13-45

SetAfPt, 27-31  
 SETDITEM, 33-357  
 SETDMENU, 33-355  
 SetDrPt, 27-29  
 SETDSUB, 33-359  
 SetOPen, 27-28  
 SETSITEM, 33-364  
 SETSMENU, 33-362  
 SETSSUB, 33-366  
 setWExcept, 33-133  
 SetWrMsk, 27-30  
 sex, 45-67  
 SGR\_BLACK, 9-37  
 SGR\_BLACKBG, 9-47  
 SGR\_BLUE, 9-41  
 SGR\_BLUEBG, 9-51  
 SGR\_BOLD, 9-31  
 SGR\_CLR0, 9-59  
 SGR\_CLR0BG, 9-68  
 SGR\_CLR1, 9-60  
 SGR\_CLR1BG, 9-69  
 SGR\_CLR2, 9-61  
 SGR\_CLR2BG, 9-70  
 SGR\_CLR3, 9-62  
 SGR\_CLR3BG, 9-71  
 SGR\_CLR4, 9-63  
 SGR\_CLR4BG, 9-72  
 SGR\_CLR5, 9-64  
 SGR\_CLR5BG, 9-73  
 SGR\_CLR6, 9-65  
 SGR\_CLR6BG, 9-74  
 SGR\_CLR7, 9-66  
 SGR\_CLR7BG, 9-75  
 SGR\_CYAN, 9-43  
 SGR\_CYANBG, 9-53  
 SGR\_DEFAULT, 9-45  
 SGR\_DEFAULTBG, 9-55  
 SGR\_GREEN, 9-39  
 SGR\_GREENBG, 9-49  
 SGR\_ITALIC, 9-32  
 SGR\_MAGENTA, 9-42  
 SGR\_MAGENTABG, 9-52  
 SGR\_NEGATIVE, 9-34  
 SGR\_PRIMARY, 9-30  
 SGR\_RED, 9-38  
 SGR\_REDBG, 9-48  
 SGR\_UNDERSCORE, 9-33  
 SGR\_WHITE, 9-44  
 SGR\_WHITEBG, 9-54  
 SGR\_YELLOW, 9-40  
 SGR\_YELLOWBG, 9-50  
 SHADE\_BW, 34-1778  
 SHADE\_COLOR, 34-1780  
 SHADE\_GREYSCALE, 34-1779  
 shape, 45-86  
 SHARED\_LOCK, 19-51

SHFCprList, 60-49  
 SHFl1st, 26-32  
 SHIFITEM, 34-1850  
 SHIFMENU, 34-1849  
 SHIFTSUB, 34-1851  
 SHIFTY, 33-327  
 SHOWTITLE, 34-1530  
 SIGBREAKB\_CTRL\_C, 19-211, 19-224  
 SIGBREAKB\_CTRL\_D, 19-212, 19-225  
 SIGBREAKB\_CTRL\_E, 19-217, 19-226  
 SIGBREAKB\_CTRL\_F, 19-218, 19-227  
 SIGBREAKB\_CTRL\_G, 19-224  
 SIGBREAKB\_CTRL\_H, 19-225  
 SIGBREAKB\_CTRL\_I, 19-226  
 SIGBREAKB\_CTRL\_J, 19-227  
 SIGN, 34-1864  
 SIGNFLAC, 3-65  
 SILENCE, 33-216  
 SIMPLE\_REFRESH, 34-1267  
 SimpleSprite, 53-10  
 sin, 42-22, 42-94  
 SINC, 42-48  
 SINGLE, 34-1750  
 sinh, 42-23, 42-94  
 SIX\_LPI, 33-283, 34-1766  
 Size, 50-32, 61-134, 61-134, 61-135, 61-135  
 SIZEBOTTOM, 34-1258  
 SIZEBRIGHT, 34-1256  
 SIZECADGET, 33-60  
 sizeof, 61-67, 61-67  
 SIZEVERIFY, 34-969  
 SIZING, 34-517  
 SKIP\_WAIT, 33-308  
 sm\_ArgList, 54-27  
 sm\_ClipID, 7-56  
 sm\_Message, 54-22  
 sm\_Msg, 7-54  
 sm\_NumArgs, 54-25  
 sm\_Process, 54-23  
 sm\_Segment, 54-24  
 sm\_ToolWindow, 54-26  
 sm\_Unit, 7-55  
 SMART\_REFRESH, 34-1266  
 SMARTCOMPILER, 61-101  
 sp\_Msg, 20-109  
 sp\_Pkt, 20-110  
 SPAbs, 43-31  
 SPACos, 43-38  
 SPAdd, 43-33  
 SPAsin, 43-38  
 SPAtan, 43-38  
 SPCmp, 43-29  
 SPCos, 43-39  
 SPCosh, 43-40  
 SPDiv, 43-36  
 SPECIAL\_ASPECT, 48-162

SPECIAL\_DENSITY1, 48-164  
 SPECIAL\_DENSITY2, 48-165  
 SPECIAL\_DENSITY3, 48-166  
 SPECIAL\_DENSITY4, 48-167  
 SPECIAL\_DENSITYMASK, 48-163  
 SPECIAL\_FRACCOLS, 48-160  
 SPECIAL\_FRACROWS, 48-161  
 SPECIAL\_FULLCOLS, 48-158  
 SPECIAL\_FULLROWS, 48-159  
 SPECIAL\_MILCOLS, 48-156  
 SPECIAL\_MILROWS, 48-157  
 SpecialInfo, 34-352  
 SpecialLink, 34-959  
 SPExp, 43-41  
 SPFeee, 43-42  
 SPFix, 43-27  
 SPFlt, 43-28  
 SPLog, 43-41  
 SPLog10, 43-41  
 SPMul, 43-35  
 SPNeg, 43-32  
 SPPow, 43-41  
 spr, 13-111  
 SprColors, 24-120  
 SprIns, 60-35  
 SPRITE\_ATTACHED, 53-8  
 SpriteDef, 13-106  
 SpriteReserved, 26-44  
 SPRITES, 60-61  
 sprpt, 13-105  
 sprRsrvd, 50-39  
 sprstop, 11-82  
 sprstrdup, 11-81  
 SPSin, 43-39  
 SPSincos, 43-39  
 SPSinh, 43-40  
 SFSqrt, 43-42  
 SPSub, 43-34  
 SPTan, 43-39  
 SPTanh, 43-40  
 SPTR, 55-36  
 SPTst, 43-30  
 sqrt, 42-24, 42-92  
 SR10, 14-25  
 SRCA, 3-53  
 SRCB, 3-52  
 SRCC, 3-51  
 SRET\_CANCELMENU, 33-352  
 SRET\_CPROP, 33-349  
 SRET\_CRELEASE, 33-347  
 SRET\_CSDRAG, 33-350  
 SRET\_CGIZING, 33-345  
 SRET\_CWDRAG, 33-346  
 SRET\_MENU, 33-344  
 SRET\_REQ, 33-348  
 SRET\_RJM, 33-342

SRET\_SMENU, 33-343  
 SRET\_STRING, 33-351  
 StandardPacket, 20-108  
 start, 11-53  
 StartMicros, 33-130  
 StartSecs, 33-130  
 stat, 3-86  
 StateReturn, 33-133  
 stderr, 55-47  
 stdin, 55-45, 55-50  
 stdout, 55-46, 55-52  
 strcmp, 61-136  
 STREQ, 61-136  
 strequ, 13-48  
 STRGADGET, 34-529  
 strhor, 13-50  
 STRINGCENTER, 34-485  
 StringInfo, 34-666  
 STRINGRIGHT, 34-487  
 strtolong, 13-51  
 STRPTR, 7-46, 56-47  
 strtol, 42-91  
 strvbl, 13-49  
 SubRect, 33-113  
 SUBDRAWN, 33-316  
 SubItem, 34-143  
 SUBNUM, 34-1847  
 SUD, 3-68  
 SUL, 3-69  
 SUPER\_BITMAP, 34-1268  
 SUPER\_UNUSED, 34-1315  
 SuperBitMap, 6-39  
 SuperClipRect, 6-40  
 SuperSaveClipRects, 6-54  
 SUPERONT, 34-521  
 SUPERONTGADGET, 33-63  
 SUSERFLAGS, 24-22  
 SwapBits, 33-398  
 SwapBitsRastPortClipRect, 33-398  
 SWE\_NOACTIVE, 33-332  
 SWE\_REQUEST, 33-331  
 SysFont, 33-174  
 SYSGADGET, 34-508  
 SysGadgets, 33-143  
 SYSREQUEST, 34-270  
 SysScreen, 33-132  
 system\_bplcon0, 26-42  
 ta\_Flags, 56-50  
 ta\_Name, 56-47  
 ta\_Style, 56-49  
 ta\_YSize, 56-48  
 tan, 42-25, 42-94  
 tanh, 42-26, 42-94  
 Task, 6-53, 20-32, 26-52, 39-33, 49-83  
 TBC\_HCLRTAB, 9-88  
 TBC\_HCLRTABSALL, 9-89

TD\_CHANGENUM, 58-82  
 TD\_CHANGESTATE, 58-83  
 TD\_FORMAT, 58-80, 58-99  
 TD\_LABELSIZE, 58-117  
 TD\_LASTCOMM, 58-86  
 TD\_MOTOR, 58-78, 58-97  
 TD\_NAME, 58-73  
 TD\_PROTSTATUS, 58-84, 58-86  
 TD\_REMOVE, 58-81  
 TD\_SECSHIFT, 58-52  
 TD\_SECTOR, 58-51  
 TD\_SEEK, 58-79, 58-98  
 TDERR\_BadDriveType, 58-140  
 TDERR\_BadHdrSum, 58-131  
 TDERR\_BadSecHdr, 58-134  
 TDERR\_BadSecID, 58-130  
 TDERR\_BadSecPreamble, 58-129  
 TDERR\_BadSecSum, 58-132  
 TDERR\_BadUnitNum, 58-139  
 TDERR\_DiskChanged, 58-136  
 TDERR\_DriveInUse, 58-141  
 TDERR\_NoMem, 58-138  
 TDERR\_NoSecHdr, 58-128  
 TDERR\_NotSpecified, 58-127  
 TDERR\_SeekError, 58-137  
 TDERR\_TooFewSecs, 58-133  
 TDERR\_WriteProt, 58-135  
 TDE\_EXTCOM, 58-75, 58-95, 58-96, 58-97, 58-98, 58-99,  
 58-100, 58-101  
 TermArray0, 52-27  
 TermArray1, 52-28  
 text, 16-20, 34-45  
 TextAttr, 16-67, 33-174, 34-750, 34-1472, 34-1567, 56-46  
 TextFont, 10-66, 16-56, 26-37, 50-74, 56-55  
 TextFonts, 26-36  
 tf\_Accessors, 56-65  
 tf\_Baseline, 56-62  
 tf\_BoldSmear, 56-63  
 tf\_CharData, 56-69  
 tf\_CharKern, 56-75  
 tf\_CharLoc, 56-72  
 tf\_CharSpace, 56-74  
 tf\_Flags, 56-60  
 tf\_HiChar, 56-68  
 tf\_LoChar, 56-67  
 tf\_Message, 56-56  
 tf\_Modulo, 56-71  
 tf\_Style, 56-59  
 tf\_XSize, 56-61  
 tf\_YSize, 56-58  
 TICKS\_PER\_SECOND, 19-70  
 Timer, 24-182, 31-14, 34-57, 49-38, 57-26  
 timerequest, 33-193, 49-81, 57-33  
 TIMERNAME, 57-26  
 TimeSet, 24-186  
 timeval, 31-133, 34-1647, 34-1649, 34-1651, 57-28, 57-35

timsrv, 26-35  
 TINY, 42-76  
 Title, 34-1115, 34-1369, 34-1461  
 TLOSS, 42-51  
 TMAalloc, 61-134  
 TmpRas, 50-29, 50-57, 50-57  
 toascii, 12-52  
 TOBB, 25-20, 25-22  
 TOF\_WaitQ, 26-53  
 TOGGLESELECT, 34-478  
 tolower, 12-51  
 ToolTypeArray, 29-41  
 top, 33-136  
 top\_layer, 39-24  
 topaz, 33-422  
 TOPAZ\_EIGHTY, 34-1627  
 TOPAZ\_SIXTY, 34-1628  
 TOPBORDER, 34-475, 34-623  
 TopEdge, 34-74, 34-118, 34-209, 34-291, 34-744,  
 34-785, 34-823, 34-1098, 34-1337, 34-1450,  
 34-1555  
 TOPHIT, 8-33  
 topmost, 50-48  
 toupper, 12-50  
 TR\_ADDRREQUEST, 57-39  
 TR\_GETSYSTIME, 57-40  
 TR\_MakeBad, 59-13  
 tr\_node, 57-34  
 TR\_NoMem, 59-12  
 TR\_NotUsed, 59-11  
 TR\_SETSYSTIME, 57-41  
 tr\_time, 57-35  
 trackdisk, 58-73  
 TRUE, 33-3, 33-18, 34-5  
 trunc, 43-23  
 tv\_micro, 57-30  
 tv\_secs, 57-29  
 TWO\_PI, 43-12  
 TxBaseline, 50-79  
 TxFlags, 50-76  
 TxHeight, 50-77  
 TxSpacing, 50-80  
 TxWidth, 50-78  
 Type, 34-1425, 34-1564, 42-36, 60-23, 61-134,  
 61-134, 61-135, 61-135  
 u1, 11-32, 11-43, 11-44  
 u2, 11-37, 11-45, 11-46  
 u3, 11-39, 11-42, 11-43, 11-44, 11-45, 11-46  
 u4, 11-38, 11-43, 11-44, 11-45, 11-46  
 UCopIns, 60-37  
 UCopList, 11-71, 11-73, 60-37  
 UCopperListInit, 27-35  
 UFB, 36-6  
 UFB\_AP, 36-23  
 UFB\_NC, 36-24  
 UFB\_NT, 36-22

UFB\_OP, 36-19  
 UFB\_RA, 36-20  
 UFB\_WA, 36-21  
 ufbfn, 36-10  
 ufbflg, 36-8  
 ufbtyp, 36-9  
 UNDERFLOW, 42-50  
 UndoBuffer, 34-672  
 UndoPos, 34-683  
 union, 11-23, 11-28, 11-33, 31-126, 49-69, 49-75  
 Unit, 7-40, 48-126, 48-140  
 UNIT\_MICROHZ, 57-23  
 UNIT\_VBLANK, 57-24  
 UPFRONTGADGET, 33-58  
 US\_LEGAL, 34-1784  
 US\_LETTER, 34-1783  
 UserData, 34-356, 34-1233, 34-1510  
 UserPort, 34-1190  
 VANILLAKEY, 34-1011  
 VBlank, 26-39  
 vbsrv, 26-35  
 vcd, 14-34  
 vcfd, 14-34  
 vcfddc, 14-34  
 vcfde, 14-34  
 vcfdi, 14-34  
 VctrPtr, 50-21  
 VctrTbl, 50-20  
 VERSIONNUMBER, 33-26  
 VertBody, 34-613  
 VertPot, 34-572  
 vposr, 13-24  
 vposv, 13-42  
 View, 26-27, 33-76, 34-33, 35-21, 35-42, 60-45  
 ViewInitX, 33-183, 34-1685  
 ViewInitY, 33-183, 34-1685  
 VIEWINITYMINREQ, 33-31  
 ViewLord, 33-76, 35-42  
 ViewModes, 34-1561  
 ViewPort, 11-61, 34-1476, 34-1476, 60-29, 60-31,  
 60-47, 60-47  
 VIEWX, 33-271  
 ViewXOffset, 34-1681  
 VIEWY, 33-272  
 ViewYOffset, 34-1683  
 VIRGINDISPLAY, 33-320  
 voice, 45-83  
 volume, 45-70  
 VP\_HIDE, 60-62  
 vposr, 13-23  
 VPOSROF, 17-38  
 vposv, 13-41  
 VPotRes, 34-621  
 VSBob, 24-122  
 VSIZEBITS, 3-16  
 VSIZEMASK, 3-18  
 VSOVERFLOW, 24-31

VSPRITE, 24-23, 24-75, 24-79, 24-80, 24-86, 24-87,  
 24-162, 24-239, 50-42  
 vthick, 33-138  
 VUserExt, 24-139  
 VUserStuff, 24-58, 24-59, 24-139  
 VWaitPos, 11-30, 11-43, 11-43  
 W\_TRACTOR, 34-1786  
 wa\_Lock, 54-31  
 wa\_Name, 54-32  
 WB\_DISKMAGIC, 61-85  
 WB\_DISKVERSION, 61-86  
 WBArg, 54-27, 54-30  
 WBDEVICE, 61-39  
 WBDISK, 61-34  
 WBDRAWER, 61-35  
 WBENCHCLOSE, 34-1083  
 WBENCHMESSAGE, 34-1003  
 WBENCHOPEN, 34-1082  
 WBENCHSCREEN, 34-1525  
 WBENCHWINDOW, 34-1310  
 WBCARBAGE, 61-38  
 WBKICK, 61-40  
 WBMessage, 33-198  
 WBOject, 29-38, 33-128, 61-61, 61-93, 61-98  
 WBorBottom, 33-159, 34-1470  
 WBorLeft, 33-156, 34-1470  
 WBorRight, 33-158, 34-1470  
 WBorTop, 33-157, 34-1470  
 wbottom, 33-141  
 WBPport, 33-197  
 WBPROJECT, 61-37  
 WBStartup, 54-21  
 WBTOOL, 61-36  
 WDOWNBACK, 34-522  
 WDRAGGING, 34-518  
 WEIRDCHO, 33-392  
 weight, 33-141  
 Width, 24-105, 33-136, 34-76, 34-120, 34-211,  
 34-293, 34-825, 34-1100, 34-1339, 34-1452,  
 34-1555, 45-84  
 Window, 6-43, 10-35, 33-78, 33-209, 34-247, 34-956,  
 34-1093, 34-1095, 34-1168, 34-1447, 35-44,  
 61-60, 61-120  
 WINDOWACTIVE, 34-1291  
 WINDOWCLOSE, 34-1253  
 WINDOWDEPTH, 34-1251  
 WINDOWDEST, 33-374  
 WINDOWDRAG, 34-1249  
 WindowPort, 34-1190  
 WINDOWREFRESH, 34-1308  
 WINDOWRESIZING, 34-1247  
 WINDOWTICKED, 34-1312  
 WLayer, 34-1241  
 wo\_Background, 61-105  
 wo\_CurrentX, 61-121  
 wo\_CurrentY, 61-122

```

wo_DefaultTool, 61-118
wo_DrawerData, 61-119
wo_DrawerOpen, 61-103
 wo_Flags, 61-108
wo_FreeList, 61-126
 wo_Gadget, 61-124
wo_IconDisp, 61-102
 wo_IconWin, 61-120
 wo_Lock, 61-130
wo_MasterNode, 61-94
 wo_Name, 61-114
wo_NameXOffset, 61-115
wo_NameYOffset, 61-116
 wo_Parent, 61-98
 wo_Selected, 61-104
 wo_SelectNode, 61-96
 wo_Siblings, 61-95
 wo_StackSize, 61-129
 wo_ToolTypes, 61-123
 wo_ToolWindow, 61-127
 wo_Type, 61-111
 wo_UseCount, 61-112
wo_UtilityNode, 61-97
wordreserved, 39-36, 50-82
wright, 33-141
WScreen, 34-1131
WUPERONT, 34-520
width, 33-141
 X, 14-26, 24-102, 34-1864, 34-1864, 34-1864,
 41-6, 41-6, 41-6, 41-6, 43-23, 43-23, 43-24,
 43-24, 53-14, 55-65, 55-65, 55-65, 55-65
 XI, 14-27
 XAccel, 24-221
 xoffset, 33-137, 34-1184, 34-1661
 XTrans, 24-199
 XVel, 24-220
 XY, 34-793
 Y, 14-26, 24-102, 53-14
 YI, 14-27
 YAccel, 24-221
 yoffset, 33-137, 34-1184, 34-1663
 YTrans, 24-199
 YVel, 24-220
 Z, 14-26
 ZI, 14-27
 _acos, 42-8
 _asin, 42-9
 _atan, 42-10
 _B, 12-30, 12-43
 _base, 55-16
 _BUFSIZ, 55-7
 _C, 12-29, 12-45
 _cbuf, 55-20
 _cliprects, 6-51
 _CopList, 11-60
 _cos, 42-11

```

```

_cosh, 42-12
_cot, 42-13
_ctype, 12-33, 12-35, 12-36, 12-37, 12-38, 12-39,
 12-40, 12-41, 12-42, 12-43, 12-44, 12-45
_exp, 42-14
_fabs, 42-15
_filbf, 55-49
_file, 55-18, 55-55
_flag, 55-17, 55-53, 55-54
_flgbf, 55-51, 55-57
_fperr, 42-85
_iob, 55-24, 55-45, 55-46, 55-47
_iobuf, 55-11, 55-24, 55-42
_IOEOF, 55-30, 55-53
_IOERR, 55-31, 55-54
_IOMYBUE, 55-29
_IONBF, 55-28
_IOREAD, 55-26
_IORW, 55-33
_IOSTRG, 55-32
_IOWRT, 55-27
 L, 12-25, 12-35, 12-37, 12-42, 12-43, 12-44
_ldexp, 42-16
_log, 42-17
_log10, 42-18
_modf, 42-19
 N, 12-26, 12-38, 12-42, 12-43, 12-44
_NEILE, 55-9, 55-24
_P, 12-28, 12-41, 12-43, 12-44
_pl, 6-56, 6-66
_p2, 6-66
_pad, 55-21
_pow, 42-20
_pow2, 42-21
_ptr, 55-13, 55-49, 55-51
_rcnt, 55-14, 55-49
_S, 12-27, 12-40
_sin, 42-22
_sinh, 42-23
_size, 55-19
_sqrt, 42-24
_tan, 42-25
_tanh, 42-26
 U, 12-24, 12-35, 12-36, 12-42, 12-43, 12-44
_ViewPort, 11-61
_wcnt, 55-15, 55-51
_X, 12-31, 12-39

```

Listing of clib/macros.h:

```
1 /* Commodore-Amiga, Inc. */
2 /* shortcuts used by c code */
3
4 #define MAX(a,b) ((a)>(b)?(a):(b))
5 #define MIN(a,b) ((a)<(b)?(a):(b))
6 #define ABS(x) ((x<0)?(-x):(x))
```

Contents

-----

```
devices/audio.h
devices/bootblock.h
devices/clipboard.h
devices/console.h
devices/conunit.h
devices/gameport.h
devices/input.h
devices/inpotevent.h
devices/keyboard.h
devices/keymap.h
devices/narrator.h
devices/parallel.h
devices/printer.h
devices/prtbase.h
devices/serial.h
devices/timer.h
devices/trackdisk.h
```

```

1 #ifndef DEVICES_AUDIO_H
2 #define DEVICES_AUDIO_H
3 /*****
4 /* Commodore-Amiga, Inc. */
5 /* audio.h */
6 /*****/
7
8 #ifndef EXEC_IO_H
9 #include "exec/io.h"
10 #endif
11
12 #define AUDIONAME "audio.device"
13
14 #define ADHARD_CHANNELS 4
15
16 #define ADALLOC_MINPREC -128
17 #define ADALLOC_MAXPREC 127
18
19 #define ADCMD_FREE (CMD_NONSTD+0)
20 #define ADCMD_SETPREC (CMD_NONSTD+1)
21 #define ADCMD_FINISH (CMD_NONSTD+2)
22 #define ADCMD_PERVOL (CMD_NONSTD+3)
23 #define ADCMD_LOCK (CMD_NONSTD+4)
24 #define ADCMD_WAITCYCLE (CMD_NONSTD+5)
25 #define ADCMDB_NOUNIT 5
26 #define ADCMDF_NOUNIT (1<<5)
27 #define ADCMD_ALLOCATE (ADCMDF_NOUNIT+0)
28
29 #define ADIOB_PERVOL 4
30 #define ADIOF_PERVOL (1<<4)
31 #define ADIOB_SYNCCYCLE 5
32 #define ADIOF_SYNCCYCLE (1<<5)
33 #define ADIOB_NOWAIT 6
34 #define ADIOF_NOWAIT (1<<6)
35 #define ADIOB_WRITEMESSAGE 7
36 #define ADIOF_WRITEMESSAGE (1<<7)
37
38 #define ADIOERR_NOALLOCATION -10
39 #define ADIOERR_ALLOCFALLED -11
40 #define ADIOERR_CHANNELSTOLEN -12
41
42 struct IOAudio {
43 struct IORequest ioa_Request;
44 WORD ioa_AllocKey;
45 UBYTE *ioa_Data;
46 ULONG ioa_Length;
47 UWORD ioa_Period;
48 UWORD ioa_Volume;
49 UWORD ioa_Cycles;
50 struct Message ioa_WriteMsg;
51 };
52
53 #endif

```

```

1 /*****/
2 /* Commodore-Amiga, Inc. */
3 /* bootblock.h */
4 /*****/
5
6 /*****/
7 *
8 * Source Control
9 * -----
10 *
11 * $Header: bootblock.h,v 27.2 85/07/10 01:55:47 neil Exp $
12 *
13 * $Locker: $
14 *
15 *****/
16
17 /***** BootBlock definition: */
18
19 struct BootBlock {
20 UBYTE bb_id[4]; /* 4 character identifier */
21 LONG bb_chksum; /* boot block checksum (balance) */
22 LONG bb_dosblock; /* reserved for DOS patch */
23 };
24
25 #define BOOTSECTS 2 /* 1K bootstrap */
26
27 #define BBID_DOS ['D', 'O', 'S', '\0']
28 #define BBID_KICK ['K', 'I', 'C', 'K']
29
30 #define BBNAME_DOS (('D'<<24)|('O'<<16)|('S'<<8))
31 #define BBNAME_KICK (('K'<<24)|('I'<<16)|('C'<<8)|('K'))
32

```



```

1 #ifndef DEVICES_CLIPBOARD_H
2 #define DEVICES_CLIPBOARD_H
3 /******
4 /* Commodore-Amiga, Inc.
5 /* clipboard.h
6 /******
7 /******
8 *
9 * clipboard device command definitions
10 *
11 *****
12
13 #ifndef EXEC_NODES_H
14 #include "exec/nodes.h"
15 #endif
16 #ifndef EXEC_LISTS_H
17 #include "exec/lists.h"
18 #endif
19 #ifndef EXEC_PORTS_H
20 #include "exec/ports.h"
21 #endif
22
23 #define CBD_POST (CMD_NONSTD+0)
24 #define CBD_CURRENTREADID (CMD_NONSTD+1)
25 #define CBD_CURRENTWRITEID (CMD_NONSTD+2)
26
27 #define CBERR_OBSOLETEID 1
28
29
30 struct ClipboardUnitPartial {
31 struct Node cu_Node; /* list of units */
32 ULONG cu_UnitNum; /* unit number for this unit */
33 /* the remaining unit data is private to the device */
34 };
35
36
37 struct IOClipReq {
38 struct Message io_Message;
39 struct Device *io_Device; /* device node pointer */
40 struct Unit *io_Unit; /* unit (driver private) */
41 UWORD io_Command; /* device command */
42 UBYTE io_Flags; /* including QUICK and SATISFY */
43 BYTE io_Error; /* error or warning num */
44 ULONG io_Actual; /* number of bytes transferred */
45 ULONG io_Length; /* number of bytes requested */
46 STRPTR io_Data; /* either clip stream or post port */
47 ULONG io_Offset; /* offset in clip stream */
48 LONG io_ClipID; /* ordinal clip identifier */
49 };
50
51 #define PRIMARY_CLIP 0 /* primary clip unit */
52
53 struct SatisfyMsg {
54 struct Message sm_Msg; /* the length will be 6 */
55 UWORD sm_Unit; /* which clip unit this is */
56 LONG sm_ClipID; /* the clip identifier of the post */
57 };
58
59 #endif

```

```

1 #ifndef DEVICES_CONSOLE_H
2 #define DEVICES_CONSOLE_H
3 /*****
4 /* Commodore-Amiga, Inc. */
5 /* console.h */
6 /*****
7 /*****
8 *
9 * Console device command definitions
10 *
11 * Source Control
12 *
13 * $Header: console.h,v 1.4 85/11/13 15:13:14 kodiak Exp $
14 *
15 * $Locker: $
16 *
17 /*****
18
19 #ifndef EXEC_IO_H
20 #include "exec/io.h"
21 #endif
22
23 /***** Console commands *****/
24 #define CD_ASKKEYMAP (CMD_NONSTD+0)
25 #define CD_SETKEYMAP (CMD_NONSTD+1)
26
27
28 /***** SGR parameters *****/
29
30 #define SGR_PRIMARY 0
31 #define SGR_BOLD 1
32 #define SGR_ITALIC 3
33 #define SGR_UNDERSCORE 4
34 #define SGR_NEGATIVE 7
35
36 /* these names refer to the ANSI standard, not the implementation */
37 #define SGR_BLACK 30
38 #define SGR_RED 31
39 #define SGR_GREEN 32
40 #define SGR_YELLOW 33
41 #define SGR_BLUE 34
42 #define SGR_MAGENTA 35
43 #define SGR_CYAN 36
44 #define SGR_WHITE 37
45 #define SGR_DEFAULT 39
46
47 #define SGR_BLACKBG 40
48 #define SGR_REDBG 41
49 #define SGR_GREENBG 42
50 #define SGR_YELLOWBG 43
51 #define SGR_BLUEBG 44
52 #define SGR_MAGENTABG 45
53 #define SGR_CYANBG 46
54 #define SGR_WHITEBG 47
55 #define SGR_DEFAULTBG 49
56
57 /* these names refer to the implementation, they are the preferred */
58 /* names for use with the Amiga console device. */
59 #define SGR_CLRO 30
60 #define SGR_CLR1 31
61 #define SGR_CLR2 32
62 #define SGR_CLR3 33
63 #define SGR_CLR4 34
64 #define SGR_CLR5 35
65 #define SGR_CLR6 36
66 #define SGR_CLR7 37
67
68 #define SGR_CLR0BG 40
69 #define SGR_CLR1BG 41
70 #define SGR_CLR2BG 42
71 #define SGR_CLR3BG 43
72 #define SGR_CLR4BG 44
73 #define SGR_CLR5BG 45
74 #define SGR_CLR6BG 46
75 #define SGR_CLR7BG 47
76
77
78 /***** DSR parameters *****/
79
80 #define DSR_CPR 6
81
82 /***** CTC parameters *****/
83 #define CTC_HSETTAB 0
84 #define CTC_HCLRTAB 2
85 #define CTC_HCLRTABSALL 5
86
87 /***** TBC parameters *****/
88 #define TBC_HCLRTAB 0
89 #define TBC_HCLRTABSALL 3
90
91 /***** SM and RM parameters *****/
92 #define M_LNM 20 /* linefeed newline mode */
93 #define M_ASM ">1" /* auto scroll mode */
94 #define M_AWM "?7" /* auto wrap mode */
95
96 #endif

```

```

1 /*****
2 /* Commodore-Amiga, Inc. */
3 /* conunit.h */
4 /*****
5 /*****
6 *
7 * Console device unit definitions
8 *
9 *****/
10
11 #ifndef EXEC_PORTS_H
12 #include "exec/ports.h"
13 #endif
14
15 #ifndef DEVICES_CONSOLE_H
16 #include "devices/console.h"
17 #endif
18
19 #ifndef DEVICES_KEYMAP_H
20 #include "devices/keymap.h"
21 #endif
22
23 #ifndef DEVICES_INPTEVENT_H
24 #include "devices/inpthevent.h"
25 #endif
26
27 #define PMB_ASM (M_LNM+1) /* internal storage bit for AS flag */
28 #define PMB_AWM (PMB_ASM+1) /* internal storage bit for AW flag */
29 #define MAXTABS 80
30
31
32 struct ConUnit {
33 struct MsgPort cu_MP;
34 /* ---- read only variables */
35 struct Window *cu_Window; /* intuition window bound to this unit */
36 WORD cu_XCP; /* character position */
37 WORD cu_YCP;
38 WORD cu_XMax; /* max character position */
39 WORD cu_YMax;
40 WORD cu_XRSize; /* character raster size */
41 WORD cu_YRSize;
42 WORD cu_XROrigin; /* raster origin */
43 WORD cu_YROrigin;
44 WORD cu_XRExtant; /* raster maxima */
45 WORD cu_YRExtant;
46 WORD cu_XMinShrink; /* smallest area intact from resize process */
47 WORD cu_YMinShrink;
48 WORD cu_XCCP; /* cursor position */
49 WORD cu_YCCP;
50
51 /* ---- read/write variables (writes must be protected) */
52 /* ---- storage for AskKeyMap and SetKeyMap */
53 struct KeyMap cu_KeyMapStruct;
54 /* ---- tab stops */
55 UWORD cu_TabStops[MAXTABS]; /* 0 at start, 0xffff at end of list */
56
57 /* ---- console rastport attributes */
58 BYTE cu_Mask;
59 BYTE cu_FgPen;
60 BYTE cu_BgPen;
61 BYTE cu_AOLPen;
62 BYTE cu_DrawMode;
63 BYTE cu_AreaPtSz;
64 APTR cu_AreaPtrn; /* cursor area pattern */
65 UBYTE cu_Minterms[8]; /* console minterms */
66 struct TextFont *cu_Font;
67 UBYTE cu_AlgoStyle;
68 UBYTE cu_TxFlags;
69 UWORD cu_TxHeight;
70 UWORD cu_TxWidth;
71 UWORD cu_TxBaseline;
72 UWORD cu_TxSpacing;
73
74 /* ---- console MODES and RAW EVENTS switches */
75 UBYTE cu_Modes[(PMB_AWM+7)/8]; /* one bit per mode */
76 UBYTE cu_RawEvents[(IECLASS_MAX+7)/8];
77 };

```

```

1 #ifndef DEVICES_GAMEPORT_H
2 #define DEVICES_GAMEPORT_H
3 /*****
4 /* Commodore-Amiga, Inc. */
5 /* gameport.h */
6 /*****
7 /*****
8 *
9 * GamePort public definitions
10 *
11 *****/
12
13 /***** GamePort commands *****/
14 #define GPD_READEVENT (CMD_NONSTD+0)
15 #define GPD_ASKCTYPE (CMD_NONSTD+1)
16 #define GPD_SETCTYPE (CMD_NONSTD+2)
17 #define GPD_ASKTRIGGER (CMD_NONSTD+3)
18 #define GPD_SETTRIGGER (CMD_NONSTD+4)
19
20 /***** GamePort structures *****/
21
22 /* gpt_Keys */
23 #define GPTB_DOWNKEYS 0
24 #define GPTF_DOWNKEYS (1<<0)
25 #define GPTB_UPKEYS 1
26 #define GPTF_UPKEYS (1<<1)
27
28 struct GamePortTrigger {
29 UWORD gpt_Keys; /* key transition triggers */
30 UWORD gpt_Timeout; /* time trigger (vertical blank units) */
31 UWORD gpt_XDelta; /* X distance trigger */
32 UWORD gpt_YDelta; /* Y distance trigger */
33 };
34
35 /***** Controller Types *****/
36 #define GPCT_ALLOCATED -1 /* allocated by another user */
37 #define GPCT_NOCONTROLLER 0
38
39 #define GPCT_MOUSE 1
40 #define GPCT_RELJOYSTICK 2
41 #define GPCT_ABSJOYSTICK 3
42
43
44 /***** Errors *****/
45 #define GPDERR_SETCTYPE 1 /* this controller not valid at this time */
46
47 #endif

```

```

1 #ifndef DEVICES_INPUT_H
2 #define DEVICES_INPUT_H
3 /*****
4 /* Commodore-Amiga, Inc. */
5 /* input.h */
6 /*****
7 /*****
8 *
9 * input device command definitions
10 *
11 *****/
12 #ifndef EXEC_IO_H
13 #include "exec/io.h"
14 #endif
15
16 #define IND_ADDHANDLER (CMD_NONSTD+0)
17 #define IND_REMHANDLER (CMD_NONSTD+1)
18 #define IND_WRITEEVENT (CMD_NONSTD+2)
19 #define IND_SETTHRESH (CMD_NONSTD+3)
20 #define IND_SETPERIOD (CMD_NONSTD+4)
21 #define IND_SETMPORT (CMD_NONSTD+5)
22 #define IND_SETMTYPE (CMD_NONSTD+6)
23 #define IND_SETMTRIG (CMD_NONSTD+7)
24
25 #endif

```

```

1 #ifndef DEVICES_INPUTEVENT_H
2 #define DEVICES_INPUTEVENT_H
3 /*-----*/
4 /* Commodore-Amiga, Inc. */
5 /* inputevent.h */
6 /*-----*/
7 /*-----*/
8 *
9 * input event definitions
10 *
11 /*-----*/
12
13 #ifndef DEVICES_TIMER_H
14 #include "devices/timer.h"
15 #endif
16
17 /*----- constants -----*/
18
19 /* --- InputEvent.ie_Class --- */
20 /* a NOP input event */
21 #define IECLASS_NULL 0x00
22 /* a raw keycode from the keyboard device */
23 #define IECLASS_RAWKEY 0x01
24 /* the raw mouse report from the game port device */
25 #define IECLASS_RAWMOUSE 0x02
26 /* a private console event */
27 #define IECLASS_EVENT 0x03
28 /* a pointer position report */
29 #define IECLASS_POINTERPOS 0x04
30 /* a timer event */
31 #define IECLASS_TIMER 0x06
32 /* select button pressed down over a gadget (address in ie_EventAddress) */
33 #define IECLASS_GADGETDOWN 0x07
34 /* select button released over the same gadget (address in ie_EventAddress) */
35 #define IECLASS_GADGETUP 0x08
36 /* some requester activity has taken place. See codes REQCLEAR and REQSET */
37 #define IECLASS_REQUESTER 0x09
38 /* this is a menu number transmission (menu number is in ie_Code) */
39 #define IECLASS_MENULIST 0x0A
40 /* user has selected the active window's close gadget */
41 #define IECLASS_CLOSEWINDOW 0x0B
42 /* this window has a new size */
43 #define IECLASS_SIZEWINDOW 0x0C
44 /* the window pointed to by ie_EventAddress needs to be refreshed */
45 #define IECLASS_REFRESHWINDOW 0x0D
46 /* new preferences are available */
47 #define IECLASS_NEWPREFS 0x0E
48 /* the disk has been removed */
49 #define IECLASS_DISKREMOVED 0x0F
50 /* the disk has been inserted */
51 #define IECLASS_DISKINSERTED 0x10
52 /* the window is about to be been made active */
53 #define IECLASS_ACTIVEWINDOW 0x11
54 /* the window is about to be made inactive */
55 #define IECLASS_INACTIVEWINDOW 0x12
56
57
58 /* the last class */
59 #define IECLASS_MAX 0x12

```

```

60
61
62
63 /* --- InputEvent.ie_Code --- */
64 /* IECLASS_RAWKEY */
65 #define IECODE_UP_PREFIX 0x80
66 #define IECODE_KEY_CODE_FIRST 0x00
67 #define IECODE_KEY_CODE_LAST 0x77
68 #define IECODE_COMM_CODE_FIRST 0x78
69 #define IECODE_COMM_CODE_LAST 0x7F
70
71 /* IECLASS_ANSI */
72 #define IECODE_C0_FIRST 0x00
73 #define IECODE_C0_LAST 0x1F
74 #define IECODE_ASCII_FIRST 0x20
75 #define IECODE_ASCII_LAST 0x7E
76 #define IECODE_ASCII_DEL 0x7F
77 #define IECODE_C1_FIRST 0x80
78 #define IECODE_C1_LAST 0x9F
79 #define IECODE_LATIN1_FIRST 0xA0
80 #define IECODE_LATIN1_LAST 0xFF
81
82 /* IECLASS_RAWMOUSE */
83 #define IECODE_LBUTTON 0x68 /* also uses IECODE_UP_PREFIX */
84 #define IECODE_RBUTTON 0x69
85 #define IECODE_MBUTTON 0x6A
86 #define IECODE_NOBUTTON 0xFF
87
88 /* IECLASS_EVENT */
89 #define IECODE_NEWACTIVE 0x01 /* active input window changed */
90
91 /* IECLASS_REQUESTER Codes */
92 /* REQSET is broadcast when the first Requester (not subsequent ones) opens
93 * in the Window
94 */
95 #define IECODE_REQSET 0x01
96 /* REQCLEAR is broadcast when the last Requester clears out of the Window */
97 #define IECODE_REQCLEAR 0x00
98
99
100 /* --- InputEvent.ie_Qualifier --- */
101 #define IEQUALIFIER_LSHIFT 0x0001
102 #define IEQUALIFIER_RSHIFT 0x0002
103 #define IEQUALIFIER_CAPSLOCK 0x0004
104 #define IEQUALIFIER_CONTROL 0x0008
105 #define IEQUALIFIER_LALT 0x0010
106 #define IEQUALIFIER_RALT 0x0020
107 #define IEQUALIFIER_LCOMMAND 0x0040
108 #define IEQUALIFIER_RCOMMAND 0x0080
109 #define IEQUALIFIER_NUMERICPAD 0x0100
110 #define IEQUALIFIER_REPEAT 0x0200
111 #define IEQUALIFIER_INTERRUPT 0x0400
112 #define IEQUALIFIER_MULTIBROADCAST 0x0800
113 #define IEQUALIFIER_LBUTTON 0x1000
114 #define IEQUALIFIER_RBUTTON 0x2000
115 #define IEQUALIFIER_MBUTTON 0x4000
116 #define IEQUALIFIER_RELATIVEMOUSE 0x8000
117
118 /*----- InputEvent -----*/
119

```

```

120 struct InputEvent {
121 struct InputEvent *ie_NextEvent; /* chronologically next event */
122 UBYTE ie_Class; /* input event class */
123 UBYTE ie_SubClass; /* optional subclass of the class */
124 UWORD ie_Code; /* input event code */
125 UWORD ie_Qualifier; /* qualifiers in effect for event*/
126 union {
127 struct {
128 WORD ie_x; /* pointer position for event*/
129 WORD ie_y;
130 } ie_xy;
131 APTR ie_addr;
132 } ie_position;
133 struct timeval ie_TimeStamp; /* system tick at event */
134 };
135
136 #define ie_X ie_position.ie_xy.ie_x
137 #define ie_Y ie_position.ie_xy.ie_y
138 #define ie_EventAddress ie_position.ie_addr
139
140 #endif

```

```

1 #ifndef DEVICES_KEYBOARD_H
2 #define DEVICES_KEYBOARD_H
3 /*****
4 /* Commodore-Amiga, Inc. */
5 /* keyboard.h */
6 /*****
7 /*****
8 *
9 * Keyboard device command definitions
10 *
11 *****/
12
13 #ifndef EXEC_IO_H
14 #include "exec/io.h"
15 #endif
16
17 #define KBD_READEVENT (CMD_NONSTD+0)
18 #define KBD_READMATRIX (CMD_NONSTD+1)
19 #define KBD_ADDRESETHANDLER (CMD_NONSTD+2)
20 #define KBD_REMRESETHANDLER (CMD_NONSTD+3)
21 #define KBD_RESETHANDLERDONE (CMD_NONSTD+4)
22
23 #endif

```

```

1 #ifndef DEVICES_KEYMAP_H
2 #define DEVICES_KEYMAP_H
3 /*****
4 /* Commodore-Amiga, Inc. */
5 /* keymap.h */
6 /*****
7 /*****
8 *
9 * console.device key map definitions
10 *
11 *****/
12
13 struct KeyMap {
14 APTR km_LoKeyMapTypes;
15 APTR km_LoKeyMap;
16 APTR km_LoCapsable;
17 APTR km_LoRepeatable;
18 APTR km_HiKeyMapTypes;
19 APTR km_HiKeyMap;
20 APTR km_HiCapsable;
21 APTR km_HiRepeatable;
22 };
23
24 #define KCB_NOP 7
25 #define KCF_NOP 0x80
26
27 #define KC_NOQUAL 0
28 #define KC_VANILLA 7 /* note that SHIFT+ALT+CTRL is VANILLA */
29 #define KCF_SHIFT 0x01
30 #define KCF_ALT 0x02
31 #define KCB_CONTROL 2
32 #define KCF_CONTROL 0x04
33 #define KCB_DOWNUP 3
34 #define KCF_DOWNUP 0x08
35
36 #define KCB_STRING 6
37 #define KCF_STRING 0x40
38
39 #endif

```

```

1 #ifndef DEVICES_NARRATOR_H
2 #define DEVICES_NARRATOR_H
3 /*****
4 /* Commodore-Amiga, Inc. */
5 /* narrator.h */
6 /*****
7 /*****
8 #ifndef EXEC_IO_H
9 #include "exec/io.h"
10 #endif
11
12 /* Error Codes */
13
14 #define ND_NoMem -2 /* Can't allocate memory */
15 #define ND_NoAudLib -3 /* Can't open audio device */
16 #define ND_MakeBad -4 /* Error in MakeLibrary call */
17 #define ND_UnitErr -5 /* Unit other than 0 */
18 #define ND_CantAlloc -6 /* Can't allocate audio channel(s) */
19 #define ND_Unimpl -7 /* Unimplemented command */
20 #define ND_NoWrite -8 /* Read for mouth without write first */
21 #define ND_Expunged -9 /* Can't open, deferred expunge bit set */
22 #define ND_PhonErr -20 /* Phoneme code spelling error */
23 #define ND_RateErr -21 /* Rate out of bounds */
24 #define ND_PitchErr -22 /* Pitch out of bounds */
25 #define ND_SexErr -23 /* Sex not valid */
26 #define ND_ModeErr -24 /* Mode not valid */
27 #define ND_FreqErr -25 /* Sampling frequency out of bounds */
28 #define ND_VolErr -26 /* Volume out of bounds */
29
30
31
32 /* Input parameters and defaults */
33
34 #define DEFPITCH 110 /* Default pitch */
35 #define DEFRATE 150 /* Default speaking rate (wpm) */
36 #define DEFVOL 64 /* Default volume (full) */
37 #define DEFREQ 22200 /* Default sampling frequency (Hz) */
38 #define MALE 0 /* Male vocal tract */
39 #define FEMALE 1 /* Female vocal tract */
40 #define NATURALFO 0 /* Natural pitch contours */
41 #define ROBOTICFO 1 /* Monotone */
42 #define DEFSEX MALE /* Default sex */
43 #define DEFMODE NATURALFO /* Default mode */
44
45
46
47 /* Parameter bounds */
48
49 #define MINRATE 40 /* Minimum speaking rate */
50 #define MAXRATE 400 /* Maximum speaking rate */
51 #define MINPITCH 65 /* Minimum pitch */
52 #define MAXPITCH 320 /* Maximum pitch */
53 #define MINFREQ 5000 /* Minimum sampling frequency */
54 #define MAXFREQ 28000 /* Maximum sampling frequency */
55 #define MINVOL 0 /* Minimum volume */
56 #define MAXVOL 64 /* Maximum volume */
57
58
59

```

```

60 /* Standard Write request */
61
62 struct narrator_rb {
63 struct IOStdReq message; /* Standard IORB */
64 UWORD rate; /* Speaking rate (words/minute) */
65 UWORD pitch; /* Baseline pitch in Hertz */
66 UWORD mode; /* Pitch mode */
67 UWORD sex; /* Sex of voice */
68 UBYTE *ch_masks; /* Pointer to audio alloc maps */
69 UWORD nm_masks; /* Number of audio alloc maps */
70 UWORD volume; /* Volume. 0 (off) thru 64 */
71 UWORD sampfreq; /* Audio sampling freq */
72 UBYTE mouths; /* If non-zero, generate mouths */
73 UBYTE chanmask; /* Which ch mask used (internal) */
74 UBYTE numchan; /* Num ch masks used (internal) */
75 UBYTE pad; /* For alignment */
76 };
77
78
79
80 /* Standard Read request */
81
82 struct mouth_rb {
83 struct narrator_rb voice; /* Speech IORB */
84 UBYTE width; /* Width (returned value) */
85 UBYTE height; /* Height (returned value) */
86 UBYTE shape; /* Internal use, do not modify */
87 UBYTE pad; /* For alignment */
88 };
89
90
91
92 #endif DEVICES_NARRATOR_H

```

```

1 /*****
2 /* Commodore-Amiga, Inc.
3 /* parallel.h
4 *****/
5
6 /*****
7 *
8 * external declarations for Parallel Port Driver
9 *
10 * SOURCE CONTROL
11 *
12 * $Header: parallel.h,v 25.0 85/03/27 19:14:15 tomp Exp $
13 * $Locker: $
14 *
15 *
16 *****/
17
18 #ifndef DEVICES_PARALLEL_H
19 #define DEVICES_PARALLEL_H
20
21 #ifndef EXEC_IO_H
22 #include "exec/io.h"
23 #endif !EXEC_IO_H
24
25 struct IOPArray {
26 ULONG PTermArray0;
27 ULONG PTermArray1;
28 };
29
30 /*****
31 /* CAUTION !! IF YOU ACCESS the parallel.device, you MUST (!!!!) use
32 an IOExtPar-sized structure or you may overlay innocent memory !! */
33 *****/
34
35 struct IOExtPar {
36 struct IOStdReq IOPar;
37
38 /* STRUCT MsgNode
39 * 0 APTR Succ
40 * 4 APTR Pred
41 * 8 UBYTE Type
42 * 9 UBYTE Pri
43 * A APTR Name
44 * E APTR ReplyPort
45 * 12 UWORD MNLength
46 * STRUCT IOExt
47 * 14 APTR io_Device
48 * 18 APTR io_Unit
49 * 1C UWORD io_Command
50 * 1E UBYTE io_Flags
51 * 1F UBYTE io_Error
52 * STRUCT IOStdExt
53 * 20 ULONG io_Actual
54 * 24 ULONG io_Length
55 * 28 APTR io_Data
56 * 2C ULONG io_Offset
57 * 30 */
58 ULONG io_PExtFlags; /* (not used) flag extension area */
59 UBYTE io_Status; /* status of parallel port and registers */

```



```

60 UBYTE io ParFlags; /* see PARFLAGS bit definitions below */
61 struct IOPArray io_PTermArray; /* termination character array */
62 };
63
64 #define PARB_SHARED 5 /* ParFlags non-exclusive access bit */
65 #define PARF_SHARED (1<<5) /* " non-exclusive access mask */
66 #define PARF_RAD_BOOGIE 3 /* " (not yet implemented) */
67 #define PARF_RAD_BOOGIE (1<<3) /* " (not yet implemented) */
68 #define PARB_EOFMODE 1 /* " EOF mode enabled bit */
69 #define PARF_EOFMODE (1<<1) /* " EOF mode enabled mask */
70 #define IOPARB_QUEUED 6 /* IO_FLAGS rqst-queued bit */
71 #define IOPARF_QUEUED (1<<6) /* " rqst-queued mask */
72 #define IOPARB_ABORT 5 /* " rqst-aborted bit */
73 #define IOPARF_ABORT (1<<5) /* " rqst-aborted mask */
74 #define IOPARB_ACTIVE 4 /* " rqst-qed-or-current bit */
75 #define IOPARF_ACTIVE (1<<4) /* " rqst-qed-or-current mask */
76 #define IOPTB_RWDIR 3 /* IO_STATUS read=0,write=1 bit */
77 #define IOPTF_RWDIR (1<<3) /* " read=0,write=1 mask */
78 #define IOPTB_PBUSY 2 /* " printer in busy toggle bit */
79 #define IOPTF_PBUSY (1<<2) /* " printer in busy toggle mask */
80 #define IOPTB_PAPEROUT 1 /* " paper out bit */
81 #define IOPTF_PAPEROUT (1<<1) /* " paper out mask */
82 #define IOPTB_PSEL 0 /* " printer selected bit */
83 #define IOPTF_PSEL (1<<0) /* " printer selected mask */
84
85 #define PARALLELNAME "parallel.device"
86
87 #define PDCMD_QUERY (CMD_NONSTD)
88 #define PDCMD_SETPARAMS (CMD_NONSTD+1)
89
90 #define ParErr_DevBusy 1
91 #define ParErr_BufTooBig 2
92 #define ParErr_InvParam 3
93 #define ParErr_LineErr 4
94 #define ParErr_NotOpen 5
95 #define ParErr_PortReset 6
96 #define ParErr_InitErr 7
97
98 #endif !DEVICES_PARALLEL_H

```

```

1 #ifndef DEVICES_PRINTER_H
2 #define DEVICES_PRINTER_H
3 /******
4 /* Commodore-Amiga, Inc.
5 /* printer.h
6 /******
7 /******
8 *
9 * printer device command definitions
10 *
11 * Source Control
12 * -----
13 * $Header: printer.h,v 1.2 85/10/09 16:16:10 kodiak Exp $
14 *
15 * $Locker: $
16 *
17 * *****/
18
19 #ifndef EXEC_NODES_H
20 #include "exec/nodes.h"
21 #endif
22
23 #ifndef EXEC_LISTS_H
24 #include "exec/lists.h"
25 #endif
26
27 #ifndef EXEC_PORTS_H
28 #include "exec/ports.h"
29 #endif
30
31 #define PRD_RAWWRITE (CMD_NONSTD+0)
32 #define PRD_PRTCOMMAND (CMD_NONSTD+1)
33 #define PRD_DUMPRPORT (CMD_NONSTD+2)
34
35 /* printer command definitions */
36
37 #define aRIS 0 /* ESCc reset ISO */
38 #define aRIN 1 /* ESC#1 initialize +++ */
39 #define aIND 2 /* ESCD lf ISO */
40 #define aNEL 3 /* ESCE return,lf ISO */
41 #define aRI 4 /* ESCM reverse lf ISO */
42
43 #define aSGR0 5 /* ESC[0m normal char set ISO */
44 #define aSGR3 6 /* ESC[3m italics on ISO */
45 #define aSGR23 7 /* ESC[23m italics off ISO */
46 #define aSGR4 8 /* ESC[4m underline on ISO */
47 #define aSGR24 9 /* ESC[24m underline off ISO */
48 #define aSGR1 10 /* ESC[lm boldface on ISO */
49 #define aSGR22 11 /* ESC[22m boldface off ISO */
50 #define aSFC 12 /* SGR30-39 set foreground color ISO */
51 #define aSBC 13 /* SGR40-49 set background color ISO */
52
53 #define aSHORP0 14 /* ESC[0w normal pitch DEC */
54 #define aSHORP2 15 /* ESC[2w elite on DEC */
55 #define aSHORP1 16 /* ESC[lw elite off DEC */
56 #define aSHORP4 17 /* ESC[4w condensed fine on DEC */
57 #define aSHORP3 18 /* ESC[3w condensed off DEC */
58 #define aSHORP6 19 /* ESC[6w enlarged on DEC */
59 #define aSHORP5 20 /* ESC[5w enlarged off DEC */

```

```

60
61 #define aDEN6 21 /* ESC[6"z shadow print on DEC (sort of) */
62 #define aDEN5 22 /* ESC[5"z shadow print off DEC */
63 #define aDEN4 23 /* ESC[4"z doublestrike on DEC */
64 #define aDEN3 24 /* ESC[3"z doublestrike off DEC */
65 #define aDEN2 25 /* ESC[2"z NLQ on DEC */
66 #define aDEN1 26 /* ESC[1"z NLQ off DEC */
67
68 #define aSUS2 27 /* ESC[2v superscript on +++ */
69 #define aSUS1 28 /* ESC[1v superscript off +++ */
70 #define aSUS4 29 /* ESC[4v subscript on +++ */
71 #define aSUS3 30 /* ESC[3v subscript off +++ */
72 #define aSUS0 31 /* ESC[0v normalize the line +++ */
73 #define aPLU 32 /* ESC[L partial line up ISO */
74 #define aPLD 33 /* ESC[K partial line down ISO */
75
76 #define aFNT0 34 /* ESC(B US char set DEC */
77 #define aFNT1 35 /* ESC(R French char set DEC */
78 #define aFNT2 36 /* ESC(K German char set DEC */
79 #define aFNT3 37 /* ESC(A UK char set DEC */
80 #define aFNT4 38 /* ESC(E Danish I char set DEC */
81 #define aFNT5 39 /* ESC(H Sweden char set DEC */
82 #define aFNT6 40 /* ESC(Y Italian char set DEC */
83 #define aFNT7 41 /* ESC(Z Spanish char set DEC */
84 #define aFNT8 42 /* ESC(J Japanese char set +++ */
85 #define aFNT9 43 /* ESC(6 Norwegian char set DEC */
86 #define aFNT10 44 /* ESC(C Danish II char set +++ */
87
88 #define aPROP2 45 /* ESC[2p proportional on +++ */
89 #define aPROP1 46 /* ESC[lp proportional off +++ */
90 #define aPROPO 47 /* ESC[0p proportional clear +++ */
91 #define aTSS 48 /* ESC[n E set proportional offset ISO */
92 #define aJFY5 49 /* ESC[5 F auto left justify ISO */
93 #define aJFY7 50 /* ESC[7 F auto right justify ISO */
94 #define aJFY6 51 /* ESC[6 F auto full justify ISO */
95 #define aJFY0 52 /* ESC[0 F auto justify off ISO */
96 #define aJFY3 53 /* ESC[3 F letter space (justify) ISO (special) */
97 #define aJFY1 54 /* ESC[1 F word fill(auto center) ISO (special) */
98
99 #define aVERP0 55 /* ESC[0z 1/8" line spacing +++ */
100 #define aVERP1 56 /* ESC[1z 1/6" line spacing +++ */
101 #define aSLPP 57 /* ESC[nt set form length n DEC */
102 #define aPERF 58 /* ESC[nq perf skip n (n>0) +++ */
103 #define aPERF0 59 /* ESC[0q perf skip off +++ */
104
105 #define aLMS 60 /* ESC#9 Left margin set +++ */
106 #define aRMS 61 /* ESC#0 Right margin set +++ */
107 #define aTMS 62 /* ESC#8 Top margin set +++ */
108 #define aBMS 63 /* ESC#2 Bottom marg set +++ */
109 #define aSTBM 64 /* ESC[Pnl;Pn2r T&B margins DEC */
110 #define aSLRM 65 /* ESC[Pnl;Pn2s L&R margin DEC */
111 #define aCAM 66 /* ESC#3 Clear margins +++ */
112
113 #define aHTS 67 /* ESC#H Set horiz tab ISO */
114 #define aVTS 68 /* ESC#J Set vertical tabs ISO */
115 #define aTBC0 69 /* ESC[0g Clr horiz tab ISO */
116 #define aTBC3 70 /* ESC[3g Clear all h tab ISO */
117 #define aTBC1 71 /* ESC[1g Clr vertical tabs ISO */
118 #define aTBC4 72 /* ESC[4g Clr all v tabs ISO */
119 #define aTBCALL 73 /* ESC#4 Clr all h & v tabs +++ */
120 #define aTBSALL 74 /* ESC#5 Set default tabs +++ */
121 #define aEXTEND 75 /* ESC[Pn"x extended commands +++ */
122
123 struct IOPrtCmdReq {
124 struct Message io_Message;
125 struct Device *io_Device; /* device node pointer */
126 struct Unit *io_Unit; /* unit (driver private) */
127 UWORD io_Command; /* device command */
128 BYTE io_Flags;
129 UWORD io_Error; /* error or warning num */
130 UWORD io_PrtCommand; /* printer command */
131 UBYTE io_Parm0; /* first command parameter */
132 UBYTE io_Parm1; /* second command parameter */
133 UBYTE io_Parm2; /* third command parameter */
134 UBYTE io_Parm3; /* fourth command parameter */
135 };
136
137 struct IODRPReq {
138 struct Message io_Message;
139 struct Device *io_Device; /* device node pointer */
140 struct Unit *io_Unit; /* unit (driver private) */
141 UWORD io_Command; /* device command */
142 UBYTE io_Flags;
143 BYTE io_Error; /* error or warning num */
144 struct RastPort *io_RastPort; /* raster port */
145 struct ColorMap *io_ColorMap; /* color map */
146 ULONG io_Modes; /* graphics viewport modes */
147 UWORD io_SrcX; /* source x origin */
148 UWORD io_SrcY; /* source y origin */
149 UWORD io_SrcWidth; /* source x width */
150 UWORD io_SrcHeight; /* source x height */
151 LONG io_DestCols; /* destination x width */
152 LONG io_DestRows; /* destination y height */
153 UWORD io_Special; /* option flags */
154 };
155
156 #define SPECIAL_MILCOLS 0x001 /* DestCols specified in 1/1000" */
157 #define SPECIAL_MILROWS 0x002 /* DestRows specified in 1/1000" */
158 #define SPECIAL_FULLCOLS 0x004 /* make DestCols maximum possible */
159 #define SPECIAL_FULLROWS 0x008 /* make DestRows maximum possible */
160 #define SPECIAL_FRACCOLS 0x010 /* DestCols is fraction of FULLCOLS */
161 #define SPECIAL_FRACROWS 0x020 /* DestRows is fraction of FULLROWS */
162 #define SPECIAL_ASPECT 0x080 /* ensure correct aspect ratio */
163 #define SPECIAL_DENSITYMASK 0xf00 /* masks out density bits */
164 #define SPECIAL_DENSITY1 0x100 /* lowest res */
165 #define SPECIAL_DENSITY2 0x200 /* next res */
166 #define SPECIAL_DENSITY3 0x300 /* next res */
167 #define SPECIAL_DENSITY4 0x400 /* highest res */
168
169 #define PDERR_CANCEL 1 /* user canceled a printer timeout */
170 #define PDERR_NOTGRAPHICS 2 /* printer cannot output graphics */
171 #define PDERR_INVERTHAM 3 /* cannot invert hold & modify print */
172 #define PDERR_BADDIMENSION 4 /* print dimensions illegal */
173 #define PDERR_DIMENSIONOVFLOW 5 /* print dimensions too large */
174 #define PDERR_INTERNALMEMORY 6 /* no memory for internal variables */
175 #define PDERR_BUFFERMEMORY 7 /* no memory for print buffer */
176 #endif

```

```

1 /*****
2 /* Commodore-Amiga, Inc. */
3 /* prtbase.h */
4 /*****
5 /*****
6 *
7 * printer device data definition
8 *
9 *****/
10
11 #ifndef DEVICES_PRTBASE_H
12 #define DEVICES_PRTBASE_H
13
14
15 #ifndef EXEC_NODES_H
16 #include "exec/nodes.h"
17 #endif
18 #ifndef EXEC_LISTS_H
19 #include "exec/lists.h"
20 #endif
21 #ifndef EXEC_PORTS_H
22 #include "exec/ports.h"
23 #endif
24 #ifndef EXEC_LIBRARIES_H
25 #include "exec/libraries.h"
26 #endif
27 #ifndef EXEC_TASKS_H
28 #include "exec/tasks.h"
29 #endif
30
31 #ifndef DEVICES_PARALLEL_H
32 #include "devices/parallel.h"
33 #endif
34 #ifndef DEVICES_SERIAL_H
35 #include "devices/serial.h"
36 #endif
37 #ifndef DEVICES_TIMER_H
38 #include "devices/timer.h"
39 #endif
40 #ifndef LIBRARIES_DOSEXTENS_I
41 #include "libraries/dosextens.h"
42 #endif
43 #ifndef INTUITION_INTUITION_H
44 #include "intuition/intuition.h"
45 #endif
46
47
48 struct DeviceData {
49 struct Library dd_Device; /* standard library node */
50 APTR dd_Segment; /* A0 when initialized */
51 APTR dd_ExecBase; /* A6 for exec */
52 APTR dd_CmdVectors; /* command table for device commands */
53 APTR dd_CmdBytes; /* bytes describing which command queue */
54 UWORD dd_NumCommands; /* the number of commands supported */
55 };
56
57
58 #define P_STKSIZE 0x800
59
60 struct PrinterData {
61 struct DeviceData pd_Device;
62 struct MsgPort pd_Unit; /* the one and only unit */
63 BPTR pd_PrinterSegment; /* the printer specific segment */
64 UWORD pd_PrinterType; /* the segment printer type */
65 struct PrinterSegment *pd_SegmentData; /* the segment data structure */
66 UBYTE *pd_PrintBuf; /* the raster print buffer */
67 VOID (*pd_PWrite)(); /* the write function */
68 VOID (*pd_PBothReady)(); /* write function's done */
69 union {
70 struct IOExtPar pd_p0;
71 struct IOExtSer pd_s0;
72 } pd_ior0;
73 #define pd_PIOR0 pd_ior0.pd_p0
74 #define pd_SIOR0 pd_ior0.pd_s0
75 union { /* and 1 for double buffering */
76 struct IOExtPar pd_p1;
77 struct IOExtSer pd_s1;
78 } pd_ior1;
79 #define pd_PIOR1 pd_ior1.pd_p1
80 #define pd_SIOR1 pd_ior1.pd_s1
81 struct timerequest pd_TIOR; /* timer I/O request */
82 struct MsgPort pd_IORPort; /* and message reply port */
83 struct Task pd_TC; /* write task */
84 UBYTE pd_Stk[P_STKSIZE]; /* and stack space */
85 UBYTE pd_Flags; /* device flags */
86 UBYTE pd_pad;
87 struct Preferences pd_Preferences; /* the latest preferences */
88 UBYTE pd_PwaitEnabled; /* wait function switch */
89 };
90
91 #define PPCB_GFX 0
92 #define PPCF_GFX 0x01
93 #define PPCB_COLOR 1
94 #define PPCF_COLOR 0x02
95
96 #define PPC_BWALPHA 0
97 #define PPC_BWGFYX 1
98 #define PPC_COLORGFYX 3
99
100 #define PCC_BW 1
101 #define PCC_YMC 2
102 #define PCC_YMC_BW 3
103 #define PCC_YMCB 4
104
105 struct PrinterExtendedData {
106 char *ped_PrinterName; /* printer name, null terminated */
107 VOID (*ped_Init)(); /* called after LoadSeg */
108 VOID (*ped_Expunge)(); /* called before UnLoadSeg */
109 VOID (*ped_Open)(); /* called at OpenDevice */
110 VOID (*ped_Close)(); /* called at CloseDevice */
111 UBYTE ped_PrinterClass; /* printer class */
112 UBYTE ped_ColorClass; /* color class */
113 UBYTE ped_MaxColumns; /* number of print columns available */
114 UBYTE ped_NumCharSets; /* number of character sets */
115 UWORD ped_NumRows; /* number of raster rows in a raster dump */
116 ULONG ped_MaxXDots; /* number of dots maximum in a raster dump */
117 ULONG ped_MaxYDots; /* number of dots maximum in a raster dump */
118 UWORD ped_XDotsInch; /* horizontal dot density */
119 UWORD ped_YDotsInch; /* vertical dot density */

```

```

120 char ***ped_Commands; /* printer text command table */
121 VOID (*ped_DoSpecial)(); /* special command handler */
122 VOID (*ped_Render)(); /* raster render function */
123 LONG ped_TimeoutSecs; /* good write timeout */
124 };
125
126 struct PrinterSegment [
127 ULONG ps_NextSegment; /* (actually a BPTR) */
128 ULONG ps_runAlert; /* MOVEQ #0,D0 : RTS */
129 UWORD ps_Version; /* segment version */
130 UWORD ps_Revision; /* segment revision */
131 struct PrinterExtendedData ps_PED; /* printer extended data */
132];
133 #endif

```

```

1 /*****
2 /* Commodore-Amiga, Inc.
3 /* serial.h
4 /*****
5 /*****
6 *
7 * external declarations for Serial Port Driver
8 *
9 * SOURCE CONTROL
10 * -----
11 * $Header: serial.h,v 25.0 85/03/27 19:14:15 tomp Exp $
12 *
13 * $Locker: $
14 *
15 *****/
16 #ifndef DEVICES_SERIAL_H
17 #define DEVICES_SERIAL_H
18
19 #ifndef EXEC_IO_H
20 #include "exec/io.h"
21 #endif !EXEC_IO_H
22
23 /* array of termination char's */
24 /* to use,see serial.doc setparams */
25
26 struct IOTArray [
27 ULONG TermArray0;
28 ULONG TermArray1;
29];
30
31 /*****/
32 /* CAUTION !! IF YOU ACCESS the serial.device, you MUST (!!!) use */
33 /* an IOExtSer-sized structure or you may overlay innocent memory !!*/
34 /*****/
35 struct IOExtSer [
36 struct IOStdReq IOSer;
37
38 /* STRUCT MsgNode
39 * 0 APTR Succ
40 * 4 APTR Pred
41 * 8 UBYTE Type
42 * 9 UBYTE Pri
43 * A APTR Name
44 * E APTR ReplyPort
45 * 12 UWORD MNLength
46 * STRUCT IOExt
47 * 14 APTR io_Device
48 * 18 APTR io_Unit
49 * 1C UWORD io_Command
50 * 1E UBYTE io_Flags
51 * 1F UBYTE io_Error
52 * STRUCT IOStdExt
53 * 20 ULONG io_Actual
54 * 24 ULONG io_Length
55 * 28 APTR io_Data
56 * 2C ULONG io_Offset
57 *
58
59 * IMPORTANT !! DON'T CHANGE the long-word alignment of ANY of these fields !!

```

```

60 * You can add to the end if you must do something.
61 * 30 */
62 ULONG io_CtlChar; /* control char's (order = xON,xOFF,INQ,ACK) */
63 ULONG io_RBufLen; /* length in bytes of serial port's read buffer */
64 ULONG io_ExtFlags; /* (not used) flag extension area */
65 ULONG io_Baud; /* baud rate requested (true baud) */
66 ULONG io_BrkTime; /* duration of break signal in MICROseconds */
67 struct IOTArray io_TermArray; /* termination character array */
68 UBYTE io_ReadLen; /* bits per read character (bit count) */
69 UBYTE io_WriteLen; /* bits per write character (bit count) */
70 UBYTE io_StopBits; /* stopbits for read (count) */
71 UBYTE io_SerFlags; /* see SerFlags bit definitions below */
72 UWORD io_Status;
73 };
74 /* status of serial port, as follows:
75 * BIT ACTIVE FUNCTION
76 * 0 low busy
77 * 1 low paper out
78 * 2 low select
79 * 3 low Data Set Ready
80 * 4 low Clear To Send
81 * 5 low Carrier Detect
82 * 6 low Ready To Send
83 * 7 low Data Terminal Ready
84 * 8 high read overrun
85 * 9 high break sent
86 * 10 high break received
87 * 11 high transmit x-OFFed
88 * 12 high receive x-OFFed
89 * 13-15 (not) reserved
90 */
91
92 #define SDCMD_QUERY CMD_NONSTD
93 #define SDCMD_BREAK (CMD_NONSTD+1)
94 #define SDCMD_SETPARAMS (CMD_NONSTD+2)
95
96 #define SERB_XDISABLED 7 /* SerFlags xOn-xOff feature disabled bit */
97 #define SERF_XDISABLED (1<<7) /* " xOn-xOff feature disabled mask */
98 #define SERB_EOFMODE 6 /* " EOF mode enabled bit */
99 #define SERF_EOFMODE (1<<6) /* " EOF mode enabled mask */
100 #define SERB_SHARED 5 /* " non-exclusive access bit */
101 #define SERF_SHARED (1<<5) /* " non-exclusive access mask */
102 #define SERB_RAD_BOOGIE 4 /* " high-speed mode active bit */
103 #define SERF_RAD_BOOGIE (1<<4) /* " high-speed mode active mask */
104 #define SERB_QUEUEDBRK 3 /* " queue this Break ioRqst */
105 #define SERF_QUEUEDBRK (1<<3) /* " queue this Break ioRqst */
106 #define SERB_7WIRE 2 /* " RS232 7-wire protocol */
107 #define SERF_7WIRE (1<<2) /* " RS232 7-wire protocol */
108 #define SERB_PARTY_ODD 1 /* " parity feature enabled bit */
109 #define SERF_PARTY_ODD (1<<1) /* " parity feature enabled mask */
110 #define SERB_PARTY_ON 0 /* " parity-enabled bit */
111 #define SERF_PARTY_ON (1<<0) /* " parity-enabled mask */
112 #define IOSERB_BUFRRREAD 7 /* io_Flags from read buffer bit */
113 #define IOSERF_BUFRRREAD (1<<7) /* " from read buffer mask */
114 #define IOSERB_QUEUED 6 /* " rqst-queued bit */
115 #define IOSERF_QUEUED (1<<6) /* " rqst-queued mask */
116 #define IOSERB_ABORT 5 /* " rqst-aborted bit */
117 #define IOSERF_ABORT (1<<5) /* " rqst-aborted mask */
118 #define IOSERB_ACTIVE 4 /* " rqst-queued-or-current bit */
119 #define IOSERF_ACTIVE (1<<4) /* " rqst-queued-or-current mask */
120 #define IOSTB_XOFFREAD 4 /* iost_hob receive currently xOFF'ed bit */
121 #define IOSTF_XOFFREAD (1<<4) /* " receive currently xOFF'ed mask */
122 #define IOSTB_XOFFWRITE 3 /* " transmit currently xOFF'ed bit */
123 #define IOSTF_XOFFWRITE (1<<3) /* " transmit currently xOFF'ed mask */
124 #define IOSTB_READBREAK 2 /* " break was latest input bit */
125 #define IOSTF_READBREAK (1<<2) /* " break was latest input mask */
126 #define IOSTB_WROTEBREAK 1 /* " break was latest output bit */
127 #define IOSTF_WROTEBREAK (1<<1) /* " break was latest output mask */
128 #define IOSTB_OVERRUN 0 /* " status word RBF overrun bit */
129 #define IOSTF_OVERRUN (1<<0) /* " status word RBF overrun mask */
130
131 #define SerErr_DevBusy 1
132 #define SerErr_BaudMismatch 2
133 #define SerErr_InvBaud 3
134 #define SerErr_BufErr 4
135 #define SerErr_InvParam 5
136 #define SerErr_LineErr 6
137 #define SerErr_NotOpen 7
138 #define SerErr_PortReset 8
139 #define SerErr_ParityErr 9
140 #define SerErr_InitErr 10
141 #define SerErr_TimerErr 11
142 #define SerErr_BufOverflow 12
143 #define SerErr_NoDSR 13
144 #define SerErr_NoCTS 14
145 #define SerErr_DetectedBreak 15
146
147 #define SERIALNAME "serial.device"
148
149 #endif !DEVICES_SERIAL_H

```

```

1 /*****
2 /* Commodore-Amiga, Inc. */
3 /* timer.h */
4 /*****
5 /*****
6 *
7 * SOURCE CONTROL
8 * -----
9 * $Header: timer.h,v 27.1 85/06/24 13:32:37 neil Exp $
10 *
11 * $Locker: $
12 *
13 *****/
14
15 #ifndef DEVICES_TIMER_H
16 #define DEVICES_TIMER_H
17
18 #ifndef EXEC_IO_H
19 #include "exec/io.h"
20 #endif EXEC_IO_H
21
22 /* unit defintions */
23 #define UNIT_MICROHZ 0
24 #define UNIT_VBLANK 1
25
26 #define TIMERNAME "timer.device"
27
28 struct timeval {
29 ULONG tv_secs;
30 ULONG tv_micro;
31 };
32
33 struct timerequest {
34 struct IORequest tr_node;
35 struct timeval tr_time;
36 };
37
38 /* IO_COMMAND to use for adding a timer */
39 #define TR_ADDREQUEST CMD_NONSTD
40 #define TR_GETSYSTIME (CMD_NONSTD+1)
41 #define TR_SETSYSTIME (CMD_NONSTD+2)
42
43 #endif DEVICES_TIMER_H

```

```

1 /*****
2 /* Commodore-Amiga, Inc. */
3 /* trackdisk.h */
4 /*****
5 /*****
6 *
7 * trackdisk.h
8 *
9 * Source Control
10 * -----
11 *
12 * $Header: trackdisk.h,v 27.3 85/07/12 23:16:05 neil Exp $
13 *
14 * $Locker: $
15 *
16 *****/
17
18
19 #ifndef DEVICES_TRACKDISK_H
20 #define DEVICES_TRACKDISK_H
21
22 #ifndef EXEC_IO_H
23 #include "exec/io.h"
24 #endif !EXEC_IO_H
25
26 /*
27 *-----
28 *
29 * Physical drive constants
30 *
31 *-----
32 */
33
34 #define NUMCYLS 80 /* normal # of cylinders */
35 #define MAXCYLS (NUMCYLS+20) /* max # cyles to look for during cal */
36 #define NUMSECS 11
37 #define NUMHEADS 2
38 #define MAXRETRY 10
39 #define NUMTRACKS (NUMCYLS*NUMHEADS)
40 #define NUMUNITS 4
41
42 /*
43 *-----
44 *
45 * Useful constants
46 *
47 *-----
48 */
49
50 /*-- sizes before mfm encoding */
51 #define TD_SECTOR 512
52 #define TD_SECSHIFT 9 /* log TD_SECTOR */
53
54 /*
55 *-----
56 *
57 * Driver Specific Commands
58 *
59 *-----

```

```

60 */
61
62 /*
63 *-- TD_NAME is a generic macro to get the name of the driver. This
64 *-- way if the name is ever changed you will pick up the change
65 *-- automatically.
66 *--
67 *-- Normal usage would be:
68 *--
69 *-- char internalName[] = TD_NAME;
70 *--
71 */
72
73 #define TD_NAME "trackdisk.device"
74
75 #define TDF_EXTCOM (1<<15) /* for internal use only! */
76
77
78 #define TD_MOTOR (CMD_NONSTD+0) /* control the disk's motor */
79 #define TD_SEEK (CMD_NONSTD+1) /* explicit seek (for testing) */
80 #define TD_FORMAT (CMD_NONSTD+2) /* format disk */
81 #define TD_REMOVE (CMD_NONSTD+3) /* notify when disk changes */
82 #define TD_CHANGENUM (CMD_NONSTD+4) /* number of disk changes */
83 #define TD_CHANGESTATE (CMD_NONSTD+5) /* is there a disk in the drive? */
84 #define TD_PROTSTATUS (CMD_NONSTD+6) /* is the disk write protected? */
85
86 #define TD_LASTCOMM TD_PROTSTATUS
87
88 /*
89 *
90 * The disk driver has an "extended command" facility. These commands
91 * take a superset of the normal IO Request block.
92 *
93 */
94
95 #define ETD_WRITE (CMD_WRITE|TDF_EXTCOM)
96 #define ETD_READ (CMD_READ|TDF_EXTCOM)
97 #define ETD_MOTOR (TD_MOTOR|TDF_EXTCOM)
98 #define ETD_SEEK (TD_SEEK|TDF_EXTCOM)
99 #define ETD_FORMAT (TD_FORMAT|TDF_EXTCOM)
100 #define ETD_UPDATE (CMD_UPDATE|TDF_EXTCOM)
101 #define ETD_CLEAR (CMD_CLEAR|TDF_EXTCOM)
102
103 /*
104 *
105 * extended IO has a larger than normal io request block.
106 *
107 */
108
109 struct IOExtTD {
110 struct IOStdReq iotd_Req;
111 ULONG iotd_Count;
112 ULONG iotd_SecLabel;
113 };
114
115 /* labels are TD_LABELSIZE bytes per sector */
116
117 #define TD_LABELSIZE 16
118
119 /*

```

```

120 *-----
121 *
122 * Driver error defines
123 *
124 *-----
125 */
126
127 #define TDERR_NotSpecified 20
128 #define TDERR_NoSecHdr 21
129 #define TDERR_BadSecPreamble 22
130 #define TDERR_BadSecID 23
131 #define TDERR_BadHdrSum 24
132 #define TDERR_BadSecSum 25
133 #define TDERR_TooFewSecs 26
134 #define TDERR_BadSecHdr 27
135 #define TDERR_WriteProt 28
136 #define TDERR_DiskChanged 29
137 #define TDERR_SeekError 30
138 #define TDERR_NoMem 31
139 #define TDERR_BadUnitNum 32
140 #define TDERR_BadDriveType 33
141 #define TDERR_DriveInUse 34
142
143 #endif DEVICES_TRACKDISK_H

```

Contents

graphics/clip.h  
graphics/collide.h  
graphics/copper.h  
graphics/display.h  
graphics/gels.h  
graphics/gfx.h  
graphics/gfxbase.h  
graphics/gfxmacros.h  
graphics/graphint.h  
graphics/layers.h  
graphics/rastport.h  
graphics/regions.h  
graphics/sprite.h  
graphics/text.h  
graphics/view.h

```
1 #ifndef GRAPHICS_CLIP_H
2 #define GRAPHICS_CLIP_H
3
4 #ifndef GRAPHICS_GFX_H
5 #include <graphics/gfx.h>
6 #endif
7 #ifndef EXEC_PORTS_H
8 #include <exec/ports.h>
9 #endif
10
11 /*****
12 /* Commodore-Amiga, Inc. */
13 /* clip.h */
14 /*****
15 /*
16 * Modification History
17 * date : author : Comments
18 * -----
19 * 02-04-85 Dale created file from graph.h
20 *****/
21
22 /* structures used by and constructed by windowlib.a */
23 /* understood by rom software */
24
25 struct Layer
26 {
27 struct Layer *front,*back; /* ignored by roms */
28 struct ClipRect *ClipRect; /* read by roms to find first cliprect */
29 struct RastPort *rp; /* ignored by roms, I hope */
30 struct Rectangle bounds; /* ignored by roms */
31 UBYTE Lock; /* roms, obey locking/unlocking
32 convention */
33 UBYTE LockCount; /* roms can nest their own locks and
34 still work */
35 UBYTE LayerLockCount; /* lock counter used by layer software */
36 UBYTE reserved;
37 UWORD reserved1;
38 UWORD Flags; /* obscured ?, Virtual BitMap? */
39 struct BitMap *SuperBitMap;
40 struct ClipRect *SuperClipRect; /* super bitmap cliprects if
41 VBitMap != 0 */
42 /* else damage cliprect list for refresh */
43 /* reserved for user interface use */
44 APTR Window;
45 SHORT Scroll_X,Scroll_Y;
46 struct MsgPort LockPort;
47 struct Message LockMessage;
48 struct MsgPort ReplyPort;
49 struct Message l_LockMessage;
50 struct Region *DamageList; /* list of rectangles to refresh
51 through */
52 struct ClipRect *_cliprects; /* system use during refresh */
53 struct Layer_Info *LayerInfo; /* points to head of the list */
54 struct Task *LayerLocker; /* points to task that has layerlock */
55 struct ClipRect *SuperSaveClipRects; /* preallocated cr's */
56 struct ClipRect *cr,*cr2,*crnew; /* used by dedice */
57 APTR _pl; /* system use, reserved */
58 };
59 struct ClipRect
```



```

60 {
61 struct ClipRect *Next; /* roms used to find next ClipRect */
62 struct ClipRect *prev; /* ignored by roms, used by windowlib */
63 struct Layer *lobs; /* ignored by roms, used by windowlib */
64 struct BitMap *BitMap;
65 struct Rectangle bounds; /* set up by windowlib, used by roms */
66 struct ClipRect *_pl,*_p2; /* system reserved */
67 LONG reserved; /* system use */
68 #ifdef NEWCLIPRECTS_1_1
69 LONG Flags; /* only exists in layer allocation */
70 #endif
71 };
72
73 /* internal cliprect flags */
74 #define CR_NEEDS_NO_CONCEALED_RASTERS 1
75
76 /* defines for code values for getcode */
77 #define ISLESSX 1
78 #define ISLESSY 2
79 #define ISGRTRX 4
80 #define ISGRTRY 8
81 #endif

```

```

1 #ifndef GRAPHICS_COLLIDE_H
2 #define GRAPHICS_COLLIDE_H
3 /*****
4 /*
5 /* Commodore-Amiga, Inc.
6 /*
7 /* Modification History
8 /* date : author : Comments
9 /* -----
10 /* 8-24-84 Dale added this header file
11 /*
12 /*****
13
14 /* include file for collision detection and control */
15
16 /* These bit descriptors are used by the GEL collide routines.
17 * These bits are set in the hitMask and meMask variables of
18 * a GEL to describe whether or not these types of collisions
19 * can affect the GEL. BNDRY_HIT is described further below;
20 * this bit is permanently assigned as the boundary-hit flag.
21 * The other bit GEL_HIT is meant only as a default to cover
22 * any GEL hitting any other; the user may redefine this bit.
23 */
24 #define BORDERHIT 0
25
26 /* These bit descriptors are used by the GEL boundry hit routines.
27 * When the user's boundry-hit routine is called (via the argument
28 * set by a call to SetCollision) the first argument passed to
29 * the user's routine is the address of the GEL involved in the
30 * boundry-hit, and the second argument has the appropriate bit(s)
31 * set to describe which boundry was surpassed
32 */
33 #define TOPHIT 1
34 #define BOTTOMHIT 2
35 #define LEFTHIT 4
36 #define RIGHTHIT 8
37
38 #endif

```

```

1 #ifndef GRAPHICS_COPPER_H
2 #define GRAPHICS_COPPER_H
3 /***** copper.h *****/
4 /*
5 Commodore-Amiga, Inc.
6 /*
7 Modification History
8 /* date : author : Comments
9 /*
10 /* 8-24-84 Dale added this header file
11 /* 9-11-84 Dale redefined with unions
12 /* 2-09-85 Dale made #defines for union ignorance
13 /*****
14
15 #define COPPER_MOVE 0 /* pseudo opcode for move #XXXX,dir */
16 #define COPPER_WAIT 1 /* pseudo opcode for wait y,x */
17 #define CPRNXTBUF 2 /* continue processing with next buffer */
18 #define CPR_NT_LOF 0x8000 /* copper instruction only for short frames */
19 #define CPR_NT_SHT 0x4000 /* copper instruction only for long frames */
20 struct CopIns
21 {
22 short OpCode; /* 0 = move, 1 = wait */
23 union
24 {
25 struct CopList *nxtlist;
26 struct
27 {
28 union
29 {
30 SHORT VWaitPos; /* vertical beam wait */
31 SHORT DestAddr; /* destination address of copper move */
32 } ul;
33 union
34 {
35 SHORT HWaitPos; /* horizontal beam wait position */
36 SHORT DestData; /* destination immediate data to send */
37 } u2;
38 } u4;
39 } u3;
40 };
41 /* shorthand for above */
42 #define NXTLIST u3.nxtlist
43 #define VWAITPOS u3.u4.ul.VWaitPos
44 #define DESTADDR u3.u4.ul.DestAddr
45 #define HWAITPOS u3.u4.u2.HWaitPos
46 #define DESTDATA u3.u4.u2.DestData
47
48
49 /* structure of cprlist that points to list that hardware actually executes */
50 struct cprlist
51 {
52 struct cprlist *Next;
53 UWORD *start; /* start of copper list */
54 SHORT max; /* number of long instructions */
55 };
56
57 struct CopList
58 {
59 struct CopList *Next; /* next block for this copper list */

```

```

60 struct CopList *_CopList; /* system use */
61 struct ViewPort *_ViewPort; /* system use */
62 struct CopIns *CopIns; /* start of this block */
63 struct CopIns *CopPtr; /* intermediate ptr */
64 UWORD *CopLStart; /* mrgcop fills this in for Long Frame*/
65 UWORD *CopSStart; /* mrgcop fills this in for Short Frame*/
66 SHORT Count; /* intermediate counter */
67 SHORT MaxCount; /* max # of copins for this block */
68 SHORT DyOffset; /* offset this copper list vertical waits */
69 };
70
71 struct UCopList
72 {
73 struct UCopList *Next;
74 struct CopList *FirstCopList; /* head node of this copper list */
75 struct CopList *CopList; /* node in use */
76 };
77
78 struct copinit
79 {
80 UWORD diagstrt[4]; /* copper list for first bitplane */
81 UWORD sprstrtup[(2*8*2)+2+(2*2)+2];
82 UWORD sprstop[2];
83 };
84
85 #endif

```

```

1 /***** display.h *****/
2 /* */
3 /* Commodore-Amiga, Inc. */
4 /* */
5 /* Modification History */
6 /* date : author : Comments */
7 /* ----- : ----- : ----- */
8 /* 8-24-84 Dale added this header file */
9 /* */
10 /*****/
11
12 /* include define file for display control registers */
13 /* bplcon0 defines */
14 #define MODE_640 0x8000
15 #define PLNCNTMSK 0x7 /* how many bit planes? */
16 /* 0 = none, 1->6 = 1->6, 7 = reserved */
17 #define PLNCNTSHFT 12 /* bits to shift for bplcon0 */
18 #define PF2PRI 0x40 /* bplcon2 bit */
19 #define COLORON 0x0200 /* disable color burst */
20 #define DBLPF 0x400
21 #define HOLDNMODIFY 0x800
22 #define INTERLACE 4 /* interlace mode for 400 */
23
24 /* bplcon1 defines */
25 #define PFA_FINE_SCROLL 0xF
26 #define PFB_FINE_SCROLL_SHIFT 4
27 #define PF_FINE_SCROLL_MASK 0xF
28
29 /* display window start and stop defines */
30 #define DIW_HORIZ_POS 0x7F /* horizontal start/stop */
31 #define DIW_VRTCL_POS 0x1FF /* vertical start/stop */
32 #define DIW_VRTCL_POS_SHIFT 7
33
34 /* Data fetch start/stop horizontal position */
35 #define DFTCH_MASK 0xFF
36
37 /* vposr bits */
38 #define VPOSRL0F 0x8000
39

```

```

1 #ifndef GRAPHICS_GELS_H
2 #define GRAPHICS_GELS_H
3
4 /*****/
5 *
6 * include file for AMIGA GELS (Graphics Elements)
7 *
8 * Commodore-Amiga, Inc.
9 *
10 * Modification History
11 * date : author : Comments
12 * ----- : ----- : -----
13 * 8-24-84 Dale added this header file
14 * 9-28-84 --RJ-- for GELS16 added Bob.h to this file
15 * made name and declaration changes
16 *
17 *****/
18
19
20 /* VSprite flags */
21 /* user-set VSprite flags: */
22 #define SUSERFLAGS 0x00FF /* mask of all user-settable VSprite-flags */
23 #define VSPRITE 0x0001 /* set if VSprite, clear if Bob */
24 #define SAVEBACK 0x0002 /* set if background is to be saved/restored */
25 #define OVERLAY 0x0004 /* set to mask image of Bob onto background */
26 #define MUSTDRAW 0x0008 /* set if VSprite absolutely must be drawn */
27 /* system-set VSprite flags: */
28 #define BACKSAVED 0x0100 /* this Bob's background has been saved */
29 #define BOBUPDATE 0x0200 /* temporary flag, useless to outside world */
30 #define GELGONE 0x0400 /* set if gel is completely clipped (offscreen) */
31 #define VSOVERFLOW 0x0800 /* VSprite overflow (if MUSTDRAW set we draw!) */
32
33 /* Bob flags */
34 /* these are the user flag bits */
35 #define BUSERFLAGS 0x00FF /* mask of all user-settable Bob-flags */
36 #define SAVEBOB 0x0001 /* set to not erase Bob */
37 #define BOBISCOMP 0x0002 /* set to identify Bob as AnimComp */
38 /* these are the system flag bits */
39 #define BWAITING 0x0100 /* set while Bob is waiting on 'after' */
40 #define BDRAWN 0x0200 /* set when Bob is drawn this DrawG pass */
41 #define BOBSAWAY 0x0400 /* set to initiate removal of Bob */
42 #define BOBNIX 0x0800 /* set when Bob is completely removed */
43 #define SAVEPRESERVE 0x1000 /* for back-restore during double-buffer */
44 #define OUTSTEP 0x2000 /* for double-clearing if double-buffer */
45
46 /* defines for the animation procedures */
47 #define ANFRACSIZE 6
48 #define ANIMHALF 0x0020
49 #define RINGSTRIGGER 0x0001
50
51
52 /* UserStuff definitions
53 * the user can define these to be a single variable or a sub-structure
54 * if undefined by the user, the system turns these into innocuous variables
55 * see the manual for a thorough definition of the UserStuff definitions
56 *
57 */
58 #ifndef VUserStuff /* VSprite user stuff */
59 #define VUserStuff SHORT

```

```

60 #endif
61
62 #ifndef BUserStuff /* Bob user stuff */
63 #define BUserStuff SHORT
64 #endif
65
66 #ifndef AUserStuff /* AnimOb user stuff */
67 #define AUserStuff SHORT
68 #endif
69
70
71
72
73 /***** GEL STRUCTURES *****/
74
75 struct VSprite
76 {
77 /* ----- SYSTEM VARIABLES ----- */
78 /* GEL linked list forward/backward pointers sorted by y,x value */
79 struct VSprite *NextVSprite;
80 struct VSprite *PrevVSprite;
81
82 /* GEL draw list constructed in the order the Bobs are actually drawn, then
83 * list is copied to clear list
84 * must be here in VSprite for system boundary detection
85 */
86 struct VSprite *DrawPath; /* pointer of overlay drawing */
87 struct VSprite *ClearPath; /* pointer for overlay clearing */
88
89 /* the VSprite positions are defined in (y,x) order to make sorting
90 * sorting easier, since (y,x) as a long integer
91 */
92 WORD OldY, OldX; /* previous position */
93
94 /* ----- COMMON VARIABLES ----- */
95 WORD Flags; /* VSprite flags */
96
97
98 /* ----- USER VARIABLES ----- */
99 /* the VSprite positions are defined in (y,x) order to make sorting
100 * sorting easier, since (y,x) as a long integer
101 */
102 WORD Y, X; /* screen position */
103
104 WORD Height;
105 WORD Width; /* number of words per row of image data */
106 WORD Depth; /* number of planes of data */
107
108 WORD MeMask; /* which types can collide with this VSprite */
109 WORD HitMask; /* which types this VSprite can collide with */
110
111 WORD *ImageData; /* pointer to VSprite image */
112
113 /* borderLine is the one-dimensional logical OR of all
114 * the VSprite bits, used for fast collision detection of edge
115 */
116 WORD *BorderLine; /* logical OR of all VSprite bits */
117 WORD *CollMask; /* similar to above except this is a matrix */
118
119 /* pointer to this VSprite's color definitions (not used by Bobs) */

```

```

120 WORD *SprColors;
121
122 struct Bob *VSBob; /* points home if this VSprite is part of
123 a Bob */
124
125 /* planePick flag: set bit selects a plane from image, clear bit selects
126 * use of shadow mask for that plane
127 * OnOff flag: if using shadow mask to fill plane, this bit (corresponding
128 * to bit in planePick) describes whether to fill with 0's or 1's
129 * There are two uses for these flags:
130 * - if this is the VSprite of a Bob, these flags describe how the Bob
131 * is to be drawn into memory
132 * - if this is a simple VSprite and the user intends on setting the
133 * MUSTDRAW flag of the VSprite, these flags must be set too to descri
134 * which color registers the user wants for the image
135 */
136 BYTE PlanePick;
137 BYTE PlaneOnOff;
138
139 VUserStuff VUserExt; /* user definable: see note above */
140 };
141
142 struct Bob
143 /* blitter-objects */
144 {
145 /* ----- SYSTEM VARIABLES ----- */
146
147 /* ----- COMMON VARIABLES ----- */
148 WORD Flags; /* general purpose flags (see definitions below) */
149
150 /* ----- USER VARIABLES ----- */
151 WORD *SaveBuffer; /* pointer to the buffer for background save */
152
153 /* used by Bobs for "cookie-cutting" and multi-plane masking */
154 WORD *ImageShadow;
155
156 /* pointer to BOBs for sequenced drawing of Bobs
157 * for correct overlaying of multiple component animations
158 */
159 struct Bob *Before; /* draw this Bob before Bob pointed to by before */
160 struct Bob *After; /* draw this Bob after Bob pointed to by after */
161
162 struct VSprite *BobVSprite; /* this Bob's VSprite definition */
163
164 struct AnimComp *BobComp; /* pointer to this Bob's AnimComp def */
165
166 struct DBufPacket *DBuffer; /* pointer to this Bob's dBuf packet */
167
168 BUserStuff BUserExt; /* Bob user extension */
169 };
170
171 struct AnimComp
172 {
173 /* ----- SYSTEM VARIABLES ----- */
174
175 /* ----- COMMON VARIABLES ----- */
176 WORD Flags; /* AnimComp flags for system & user */
177
178 /* timer defines how long to keep this component active:
179 * if set non-zero, timer decrements to zero then switches to nextSeq

```

```

180 * if set to zero, AnimComp never switches
181 */
182 WORD Timer;
183
184 /* ----- USER VARIABLES ----- */
185 /* initial value for timer when the AnimComp is activated by the system */
186 WORD TimeSet;
187
188 /* pointer to next and previous components of animation object */
189 struct AnimComp *NextComp;
190 struct AnimComp *PrevComp;
191
192 /* pointer to component definition of next image in sequence */
193 struct AnimComp *NextSeq;
194 struct AnimComp *PrevSeq;
195
196 WORD (*AnimCRoutine)(); /* address of special animation procedure */
197
198 WORD YTrans; /* initial y translation (if this is a component) */
199 WORD XTrans; /* initial x translation (if this is a component) */
200
201 struct AnimOb *HeadOb;
202
203 struct Bob *AnimBob;
204];
205
206 struct AnimOb
207 {
208 /* ----- SYSTEM VARIABLES ----- */
209 struct AnimOb *NextOb, *PrevOb;
210
211 /* number of calls to Animate this AnimOb has endured */
212 LONG Clock;
213
214 WORD AnOldY, AnOldX; /* old y,x coordinates */
215
216 /* ----- COMMON VARIABLES ----- */
217 WORD AnY, AnX; /* y,x coordinates of the AnimOb */
218
219 /* ----- USER VARIABLES ----- */
220 WORD YVel, XVel; /* velocities of this object */
221 WORD YAccel, XAccel; /* accelerations of this object */
222
223 WORD RingYTrans, RingXTrans; /* ring translation values */
224
225 WORD (*AnimORoutine)(); /* address of special animation
226 procedure */
227
228 struct AnimComp *HeadComp; /* pointer to first component */
229
230 AUserStuff AUserExt; /* AnimOb user extension */
231 };
232
233 /* dBufPacket defines the values needed to be saved across buffer to buffer
234 * when in double-buffer mode
235 */
236 struct DBufPacket
237 {
238 WORD BufY, BufX; /* save the other buffers screen coordinates */
239 struct VSprite *BufPath; /* carry the draw path over the gap */

```

```

240
241 /* these pointers must be filled in by the user */
242 /* pointer to other buffer's background save buffer */
243 WORD *BufBuffer;
244 };
245
246
247
248 /* *****
249
250 /* these are GEL functions that are currently simple enough to exist as a
251 * definition. It should not be assumed that this will always be the case
252 */
253 #define InitAnimate(animKey) {*(animKey) = NULL;}
254 #define RemBob(b) {(b)->Flags |= BOBSAWAY;}
255
256
257 /* *****
258
259 #define B2NORM 0
260 #define B2SWAP 1
261 #define B2BOBBER 2
262
263 /* *****
264
265 /* a structure to contain the 16 collision procedure addresses */
266 struct collTable
267 {
268 int (*collPtrs[16])();
269 };
270
271
272 #endif

```

```

1 #ifndef GRAPHICS_GFX_H
2 #define GRAPHICS_GFX_H
3 /***** gfx.h *****/
4 /*
5 /* Commodore-Amiga, Inc.
6 /*
7 /* Modification History
8 /* date :author :Comments
9 /*-----*/
10 /* 8-24-84 Dale added this header file
11 /* Feb 85 Dale added Rectangle, BitMap structures
12 /*****
13
14 /* general include file for application programs */
15 #define BITSET 0x8000
16 #define BITCLR 0
17
18 #define AGNUS
19 #ifdef AGNUS
20 #define TOBB(a) ((long)(a))
21 #else
22 #define TOBB(a) ((long)(a)>>1) /* convert Chip adr to Bread Board Adr */
23 #endif
24
25 struct Rectangle
26 {
27 SHORT MinX,MinY;
28 SHORT MaxX,MaxY;
29 };
30
31 typedef UBYTE *PLANEPTR;
32
33 struct BitMap
34 {
35 UWORD BytesPerRow;
36 UWORD Rows;
37 UBYTE Flags;
38 UBYTE Depth;
39 UWORD pad;
40 PLANEPTR Planes[8];
41 };
42
43 #define RASSIZE(w,h) ((h)*(w+15)>>3&0xFFFE)
44
45 #endif

```

```

1 #ifndef GRAPHICS_GFXBASE_H
2 #define GRAPHICS_GFXBASE_H
3
4 #ifndef EXEC_LISTS_H
5 #include <exec/lists.h>
6 #endif
7 #ifndef EXEC_LIBRARIES_H
8 #include <exec/libraries.h>
9 #endif
10 #ifndef EXEC_INTERRUPTS_H
11 #include <exec/interrupts.h>
12 #endif
13
14 /***** gfxbase.h *****/
15 /*
16 /* Commodore-Amiga, Inc.
17 /*
18 /* Modification History
19 /* date : author : Comments
20 /*-----*/
21 /* 10-20-84 Kodiak added this header file & TextFonts
22 /*
23 /*****
24 struct GfxBase
25 {
26 struct Library LibNode;
27 struct View *ActiView;
28 struct copinit *copinit; /* ptr to copper start up list */
29 long *cia; /* for 8520 resource use */
30 long *blitter; /* for future blitter resource use */
31 UWORD *LOFlist;
32 UWORD *SHFlist;
33 struct bltnode *blthd,*blttl;
34 struct bltnode *bsblthd,*bsblttl;
35 struct Interrupt vbsrv,timsrv,bltsrv;
36 struct List TextFonts;
37 struct TextFont *DefaultFont;
38 UWORD Modes; /* copy of current first bplcon0 */
39 BYTE VBlank;
40 BYTE Debug;
41 SHORT BeamSync;
42 SHORT system_bplcon0; /* this is initialized to 0 */
43 /* it is ored into each bplcon0 for display */
44 UBYTE SpriteReserved;
45 UBYTE bytesreserved;
46 /* candidates for removal */
47 USHORT Flags;
48 SHORT BlitLock;
49 short BlitNest;
50
51 struct List BlitWaitQ;
52 struct Task *BlitOwner;
53 struct List TOF_WaitQ;
54 UWORD DisplayFlags; /* NTSC PAL GENLOC etc*/
55 /* Display flags are determined at power on */
56 ULONG reserved[2]; /* for future use */
57 };
58
59 #define NTSC 1

```

```

60 #define GENLOC 2
61 #define PAL 4
62
63 #define BLITMSG_FAULT 4
64 #endif

```

```

1 #ifndef GRAPHICS_GFXMACROS_H
2 #define GRAPHICS_GFXMACROS_H
3 /***** gfxmacros.h *****/
4 /*
5 /* Commodore-Amiga, Inc.
6 /*
7 /* Modification History
8 /* date : author : Comments
9 /* -----
10 /* 8-24-84 Dale added this header file
11 /* 9-06-84 Dale fixed macros using w-> to use (w)->
12 /* 9-07-84 Dale fixed macros to use new RastPort
13 /*
14 /*****/
15
16 #ifndef GRAPHICS_RASTPORT_H
17 #include <graphics/rastport.h>
18 #endif
19
20 #define ON_DISPLAY custom.dmacon = BITSET|DMAF_RASTER;
21 #define OFF_DISPLAY custom.dmacon = BITCLR|DMAF_RASTER;
22 #define ON_SPRITE custom.dmacon = BITSET|DMAF_SPRITE;
23 #define OFF_SPRITE custom.dmacon = BITCLR|DMAF_SPRITE;
24
25 #define ON_VBLANK custom.intena = BITSET|INTF_VERTB
26 #define OFF_VBLANK custom.intena = BITCLR|INTF_VERTB
27
28 #define SetOPen(w,c) [(w)->AOLPen = c;(w)->Flags |= AREAOUTLINE;}
29 #define SetDrPt(w,p) [(w)->LinePtrn = p;(w)->Flags |= FRST_DOT;}
30 #define SetWrMsk(w,m) [(w)->Mask = m;}
31 #define SetAfPt(w,p,n) [(w)->AreaPtrn = p;(w)->AreaPtSz = n;}
32
33 #define BNDRYOFF (w) [->Flags &= ~AREAOUTLINE]
34
35 #define CINIT(c,n) [UCopperListInit(c,n);]
36 #define CMOVE(c,a,b) [CMove(c,&a,b);CBump(c);]
37 #define CWAIT(c,a,b) [CWait(c,a,b);CBump(c);]
38 #define CEND(c) [CWAIT(c,10000,255);]
39
40 #endif

```

```

1 #ifndef GRAPHICS_GRAPHINT_H
2 #define GRAPHICS_GRAPHINT_H
3 /*****
4 /* Commodore-Amiga, Inc. */
5 /* graphint.h */
6 /*****/
7
8 #ifndef EXEC_NODES_H
9 #include <exec/nodes.h>
10 #endif
11
12 /* structure used by AddTOFTask */
13 struct Isrvstr
14 {
15 struct Node is_Node;
16 struct Isrvstr *Iptr; /* passed to srvr by os */
17 int (*code)();
18 int (*ccode)();
19 int Carg;
20 };
21
22 #endif

```

```

1 /*****
2 /* Commodore-Amiga, Inc. */
3 /*****/
4
5 #ifndef GRAPHICS_LAYERS_H
6 #define GRAPHICS_LAYERS_H
7
8 #ifndef EXEC_PORTS_H
9 #include <exec/ports.h>
10 #endif
11
12 #ifndef EXEC_LISTS_H
13 #include <exec/lists.h>
14 #endif
15
16 #define LAYERSIMPLE 1
17 #define LAYERSMART 2
18 #define LAYERSUPER 4
19 #define LAYERBACKDROP 0x40
20 #define LAYERREFRESH 0x80
21
22 struct Layer_Info
23 {
24 struct Layer *top_layer;
25 struct Layer *check_lp; /* system use */
26 struct Layer *obs; /* system use */
27 struct MsgPort RP_ReplyPort; /* for rastport locking */
28 struct MsgPort LockPort; /* for screen locking */
29 UBYTE Lock;
30 UBYTE broadcast; /* bunch of messages sent */
31 UBYTE LockNest;
32 UBYTE Flags;
33 struct Task *Locker;
34 BYTE fatten_count;
35 UBYTE bytereserved;
36 UWORD wordreserved; /* used to be a node in here someplace */
37 UWORD LayerInfo_extra_size;
38 ULONG longreserved;
39 struct LayerInfo_extra *LayerInfo_extra;
40 };
41
42 #define NEWLAYERINFO_CALLED 1
43 #define ALERTLAYERSNOMEM 0x83010000
44
45 #endif

```



```

1 #ifndef GRAPHICS_RASTPORT_H
2 #define GRAPHICS_RASTPORT_H
3
4 #ifndef GRAPHICS_GFX_H
5 #include <graphics/gfx.h>
6 #endif
7
8 /***** rastport.h *****/
9 *
10 * Commodore-Amiga, Inc.
11 *
12 * Modification History
13 * date : author : Comments
14 * -----
15 * 02-04-85 Dale created from graph.h
16 *****/
17
18 struct AreaInfo
19 {
20 SHORT *VctrTbl; /* ptr to start of vector table */
21 SHORT *VctrPtr; /* ptr to current vertex */
22 BYTE *FlagTbl; /* ptr to start of vector flag table */
23 BYTE *FlagPtr; /* ptrs to areafill flags */
24 SHORT Count; /* number of vertices in list */
25 SHORT MaxCount; /* AreaMove/Draw will not allow Count>MaxCount*/
26 SHORT FirstX,FirstY; /* first point for this polygon */
27 };
28
29 struct TmpRas
30 {
31 BYTE *RasPtr;
32 LONG Size;
33 /* other misc junk for freelist etc. */
34 };
35
36 /* unoptimized for 32bit alignment of pointers */
37 struct GelsInfo
38 {
39 BYTE sprRsrvd; /* flag of which sprites to reserve from
40 vsprite system */
41 UBYTE Flags; /* system use */
42 struct VSprite *gelHead, *gelTail; /* dummy vSprites for list management*/
43 /* pointer to array of 8 WORDS for sprite available lines */
44 WORD *nextLine;
45 /* pointer to array of 8 pointers for color-last-assigned to vSprites */
46 WORD **lastColor;
47 struct collTable *collHandler; /* addresses of collision routines */
48 short leftmost, rightmost, topmost, bottommost;
49 APTR firstBlissObj, lastBlissObj; /* system use only */
50 };
51
52 struct RastPort
53 {
54 struct Layer *Layer;
55 struct BitMap *BitMap;
56 USHORT *AreaPtrn; /* ptr to areafill pattern */
57 struct TmpRas *TmpRas;
58 struct AreaInfo *AreaInfo;
59 struct GelsInfo *GelsInfo;
60 UBYTE Mask; /* write mask for this raster */
61 BYTE FgPen; /* foreground pen for this raster */
62 BYTE BgPen; /* background pen */
63 BYTE AOlPen; /* areafill outline pen */
64 BYTE DrawMode; /* drawing mode for fill, lines, and text */
65 BYTE AreaPtSz; /* 2^n words for areafill pattern */
66 BYTE linpatcnt; /* current line drawing pattern preshift */
67 BYTE dummy;
68 USHORT Flags; /* miscellaneous control bits */
69 USHORT LinePtrn; /* 16 bits for textured lines */
70 SHORT cp_x, cp_y; /* current pen position */
71 UBYTE minterms[8];
72 SHORT PenWidth;
73 SHORT PenHeight;
74 struct TextFont *Font; /* current font address */
75 UBYTE AlgoStyle; /* the algorithmically generated style */
76 UBYTE TxFlags; /* text specific flags */
77 UWORD TxHeight; /* text height */
78 UWORD TxWidth; /* text nominal width */
79 UWORD TxBaseline; /* text baseline */
80 WORD TxSpacing; /* text spacing (per character) */
81 APTR *RP_User;
82 UWORD wordreserved[7]; /* used to be a node */
83 ULONG longreserved[2];
84 UBYTE reserved[8]; /* for future use */
85 };
86
87 /* drawing modes */
88 #define JAM1 0 /* jam 1 color into raster */
89 #define JAM2 1 /* jam 2 colors into raster */
90 #define COMPLEMENT 2 /* XOR bits into raster */
91 #define INVERSVID 4 /* inverse video for drawing modes */
92
93 /* these are the flag bits for RastPort flags */
94 #define FRST_DOT 0x01 /* draw the first dot of this line ? */
95 #define ONE_DOT 0x02 /* use one dot mode for drawing lines */
96 #define DBUFFER 0x04 /* flag set when RastPorts
97 are double-buffered */
98
99 /* only used for bobs */
100
101 #define AREAOUTLINE 0x08 /* used by areafiller */
102 #define NOCROSSFILL 0x20 /* areafills have no crossovers */
103
104 /* there is only one style of clipping: raster clipping */
105 /* this preserves the continuity of jaggies regardless of clip window */
106 /* When drawing into a RastPort, if the ptr to ClipRect is nil then there */
107 /* is no clipping done, this is dangerous but useful for speed */
108
109 #endif

```

```

1 #ifndef GRAPHICS_REGIONS_H
2 #define GRAPHICS_REGIONS_H
3
4 #ifndef GRAPHICS_GFX_H
5 #include <graphics/gfx.h>
6 #endif
7 /******
8 /* Commodore-Amiga, Inc. */
9 /* regions.h */
10 /******
11
12 struct RegionRectangle
13 {
14 struct RegionRectangle *Next,*Prev;
15 struct Rectangle bounds;
16 };
17
18 struct Region
19 {
20 struct Rectangle bounds;
21 struct RegionRectangle *RegionRectangle;
22 };
23
24 #endif

```

```

1 #ifndef GRAPHICS_SPRITE_H
2 #define GRAPHICS_SPRITE_H
3 /******
4 /* Commodore-Amiga, Inc. */
5 /* sprite.h */
6 /******
7
8 #define SPRITE_ATTACHED 0x80
9
10 struct SimpleSprite
11 {
12 UWORD *posctldata;
13 UWORD height;
14 UWORD x,y; /* current position */
15 UWORD num;
16 };
17 #endif

```

```

1 #ifndef GRAPHICS_TEXT_H
2 #define GRAPHICS_TEXT_H
3 /*****
4 /* Commodore-Amiga, Inc. */
5 /* file.h */
6 /*****
7 /*****
8 * graphics library text structures
9 *
10 *****/
11
12 #ifndef EXEC_PORTS_H
13 #include "exec/ports.h"
14 #endif
15
16 /*----- Font Styles -----*/
17 #define FS_NORMAL 0 /* normal text (no style bits set) */
18 #define FSB_EXTENDED 3 /* extended face (wider than normal) */
19 #define FSF_EXTENDED (1<<3)
20 #define FSB_ITALIC 2 /* italic (slanted 1:2 right) */
21 #define FSF_ITALIC (1<<2)
22 #define FSB_BOLD 1 /* bold face text (ORed w/ shifted) */
23 #define FSF_BOLD (1<<1)
24 #define FSB_UNDERLINED 0 /* underlined (under baseline) */
25 #define FSF_UNDERLINED (1<<0)
26
27 /*----- Font Flags -----*/
28 #define FPB_ROMFONT 0 /* font is in rom */
29 #define PPF_ROMFONT (1<<0)
30 #define FPB_DISKFONT 1 /* font is from diskfont.library */
31 #define PPF_DISKFONT (1<<1)
32 #define FPB_REVPATH 2 /* designed path is reversed (e.g. left) */
33 #define PPF_REVPATH (1<<2)
34 #define FPB_TALLDOT 3 /* designed for hires non-interlaced */
35 #define PPF_TALLDOT (1<<3)
36 #define FPB_WIDEDOT 4 /* designed for lores interlaced */
37 #define PPF_WIDEDOT (1<<4)
38 #define FPB_PROPORTIONAL 5 /* character sizes can vary from nominal */
39 #define PPF_PROPORTIONAL (1<<5)
40 #define FPB_DESIGNED 6 /* size is "designed", not constructed */
41 #define PPF_DESIGNED (1<<6)
42 #define FPB_REMOVED 7 /* the font has been removed */
43 #define PPF_REMOVED (1<<7)
44
45 /***** TextAttr node, matches text attributes in RastPort *****/
46 struct TextAttr {
47 STRPTR ta_Name; /* name of the font */
48 UWORD ta_YSize; /* height of the font */
49 UBYTE ta_Style; /* intrinsic font style */
50 UBYTE ta_Flags; /* font preferences and flags */
51 };
52
53
54 /***** TextFonts node *****/
55 struct TextFont {
56 struct Message tf_Message; /* reply message for font removal */
57 UWORD tf_YSize; /* font name in LN \ used in this */
58 UBYTE tf_Style; /* font height | order to best */
59 UBYTE tf_Style; /* font style | match a font */

```

```

60 UBYTE tf_Flags; /* preferences and flags / request. */
61 UWORD tf_XSize; /* nominal font width */
62 UWORD tf_Baseline; /* distance from the top of char to baseline */
63 UWORD tf_BoldSmear; /* smear to affect a bold enhancement */
64
65 UWORD tf_Accessors; /* access count */
66
67 UBYTE tf_LoChar; /* the first character described here */
68 UBYTE tf_HiChar; /* the last character described here */
69 APTR tf_CharData; /* the bit character data */
70
71 UWORD tf_Modulo; /* the row modulo for the strike font data */
72 APTR tf_CharLoc; /* ptr to location data for the strike font */
73 /* 2 words: bit offset then size */
74 APTR tf_CharSpace; /* ptr to words of proportional spacing data */
75 APTR tf_Chkern; /* ptr to words of kerning data */
76 };
77
78 #endif

```

```

1 #ifndef GRAPHICS_VIEW_H
2 #define GRAPHICS_VIEW_H
3
4 #ifndef GRAPHICS_GFX_H
5 #include <graphics/gfx.h>
6 #endif
7
8 /*****
9 /* Commodore-Amiga, Inc.
10 /* view.h
11 /*****
12 /*****
13 * Modification History
14 * date : author : Comments
15 * -----
16 * 2-4-85 Dale created from graph.h
17 * 2-8-85 Dale conversion to 24 View->ViewPort
18 *****/
19
20 struct ColorMap
21 {
22 UBYTE Flags;
23 UBYTE Type;
24 UWORD Count;
25 APTR ColorTable;
26 };
27 /* if Type == 0 then ColorTable is a table of UWORDS xRGB */
28
29 struct ViewPort
30 {
31 struct ViewPort *Next;
32 struct ColorMap *ColorMap; /* table of colors for this viewport */
33 /* if this is nil, MakeVPort assumes default values */
34 struct CopList *DspIns; /* user by MakeView() */
35 struct CopList *SprIns; /* used by sprite stuff */
36 struct CopList *ClrIns; /* used by sprite stuff */
37 struct UCopList *UCopIns; /* User copper list */
38 SHORT DWidth,DHeight;
39 SHORT DxOffset,DyOffset;
40 UWORD Modes;
41 UWORD reserved;
42 struct RasInfo *RasInfo;
43 };
44
45 struct View
46 {
47 struct ViewPort *ViewPort;
48 struct cprlist *IOFCprList; /* used for interlaced and noninterlaced */
49 struct cprlist *SHFCprList; /* only used during interlace */
50 short DyOffset,DxOffset; /* for complete View positioning */
51 /* offsets are +- adjustments to standard #s */
52 UWORD Modes; /* such as INTERLACE, GENLOC */
53 };
54
55 /* defines used for Modes in IVPargs */
56 #define PFBA 0x40
57 #define DUALPF 0x400
58 #define HIRES 0x8000
59 #define LACE 4

```

```

60 #define HAM 0x800
61 #define SPRITES 0x4000 /* reuse one of plane ctr bits */
62 #define VP_HIDE 0x2000 /* reuse another plane crt bit */
63 #define GENLOCK_AUDIO 0x100
64 #define GENLOCK_VIDEO 2
65 #define EXTRA_HALFBRITE 0x80
66
67 struct RasInfo /* used by callers to and InitDspC() */
68 {
69 struct RasInfo *Next; /* used for dualpf */
70 struct BitMap *BitMap;
71 SHORT RxOffset,RyOffset; /* scroll offsets in this BitMap */
72 };
73
74 #endif

```

Contents

-----

hardware/adkbits.h  
hardware/blit.h  
hardware/cia.h  
hardware/custom.h  
hardware/dmabits.h  
hardware/intbits.h

D -140

```
1 /*****
2 * Commodore-Amiga, Inc.
3 * adkbits.h -- bit definitions for adkcon register
4 *
5 * $Header: adkbits.h,v 27.1 85/06/24 14:42:34 neil Exp $
6 *
7 * $Locker: $
8 *
9 *****/
10
11 #ifndef HARDWARE_ADKBITS_H
12 #define HARDWARE_ADKBITS_H
13
14 #define ADKB_SETCLR 15 /* standard set/clear bit */
15 #define ADKB_PRECOMP1 14 /* two bits of precompensation */
16 #define ADKB_PRECOMP0 13
17 #define ADKB_MFMPREC 12 /* use mfm style precompensation */
18 #define ADKB_UARTBRK 11 /* force uart output to zero */
19 #define ADKB_WORDSYNC 10 /* enable DSKSYNC register matching */
20 #define ADKB_MSBSYNC 9 /* (Apple GCR Only) sync on MSB for reading */
21 #define ADKB_FAST 8 /* 1 -> 2 us/bit (mfm), 2 -> 4 us/bit (gcr) */
22 #define ADKB_USE3PN 7 /* use aud chan 3 to modulate period of ?? */
23 #define ADKB_USE2P3 6 /* use aud chan 2 to modulate period of 3 */
24 #define ADKB_USE1P2 5 /* use aud chan 1 to modulate period of 2 */
25 #define ADKB_USE0P1 4 /* use aud chan 0 to modulate period of 1 */
26 #define ADKB_USE3VN 3 /* use aud chan 3 to modulate volume of ?? */
27 #define ADKB_USE2V3 2 /* use aud chan 2 to modulate volume of 3 */
28 #define ADKB_USE1V2 1 /* use aud chan 1 to modulate volume of 2 */
29 #define ADKB_USE0V1 0 /* use aud chan 0 to modulate volume of 1 */
30
31 #define ADKE_SETCLR (1<<15)
32 #define ADKE_PRECOMP1 (1<<14)
33 #define ADKE_PRECOMP0 (1<<13)
34 #define ADKE_MFMPREC (1<<12)
35 #define ADKE_UARTBRK (1<<11)
36 #define ADKE_WORDSYNC (1<<10)
37 #define ADKE_MSBSYNC (1<<9)
38 #define ADKE_FAST (1<<8)
39 #define ADKE_USE3PN (1<<7)
40 #define ADKE_USE2P3 (1<<6)
41 #define ADKE_USE1P2 (1<<5)
42 #define ADKE_USE0P1 (1<<4)
43 #define ADKE_USE3VN (1<<3)
44 #define ADKE_USE2V3 (1<<2)
45 #define ADKE_USE1V2 (1<<1)
46 #define ADKE_USE0V1 (1<<0)
47
48 #define ADKE_PRE000NS 0 /* 000 ns of precomp */
49 #define ADKE_PRE140NS (ADKE_PRECOMP0) /* 140 ns of precomp */
50 #define ADKE_PRE280NS (ADKE_PRECOMP1) /* 280 ns of precomp */
51 #define ADKE_PRE560NS (ADKE_PRECOMP0|ADKE_PRECOMP1) /* 560 ns of precomp */
52
53 #endif !HARDWARE_ADKBITS_H
```

```

1 /*****
2 * blit.h
3 * Commodore-Amiga, Inc.
4 *
5 * $Header: blit.h,v 27.1 85/06/24 14:42:40 neil Exp $
6 *
7 * $Locker: $
8 *
9 *****/
10
11 #ifndef HARDWARE_BLIT_H
12 #define HARDWARE_BLIT_H
13
14 /* include file for blitter */
15 #define HSIZEBITS 6
16 #define VSIZEBITS 16-HSIZEBITS
17 #define HSIZEMASK 0x3f /* 2^6 -- 1 */
18 #define VSIZEMASK 0x3FF /* 2^10 - 1 */
19
20 #define MAXBYTESPERROW 128
21
22 /* definitions for blitter control register 0 */
23
24 #define ABC 0x80
25 #define ABNC 0x40
26 #define ANBC 0x20
27 #define ANBNC 0x10
28 #define NABC 0x8
29 #define NABNC 0x4
30 #define NANBC 0x2
31 #define NANBNC 0x1
32
33 /* some commonly used operations */
34 #define A_OR_B ABC|ANBC|NABC | ABNC|ANBNC|NABNC
35 #define A_OR_C ABC|NABC|ABNC | ANBC|NANBC|ANBNC
36 #define A_XOR_C NABC|ABNC | NANBC|ANBNC
37 #define A_TO_D ABC|ANBC|ABNC|ANBNC
38
39 #define BCOB_DEST 8
40 #define BCOB_SRCC 9
41 #define BCOB_SRCB 10
42 #define BCOB_SRC_A 11
43 #define BCOF_DEST 0x100
44 #define BCOF_SRCC 0x200
45 #define BCOF_SRCB 0x400
46 #define BCOF_SRC_A 0x800
47
48 #define BCLF_DESC 2 /* blitter descend direction */
49
50 #define DEST 0x100
51 #define SRCC 0x200
52 #define SRCB 0x400
53 #define SRC_A 0x800
54
55 #define ASHIFTSHIFT 12 /* bits to right align ashift value */
56 #define BSHIFTSHIFT 12 /* bits to right align bshift value */
57
58 /* definitions for blitter control register 1 */
59 #define LINEMODE 0x1
60 #define FILL_OR 0x8
61 #define FILL_XOR 0x10
62 #define FILL_CARRYIN 0x4
63 #define ONEDOT 0x2 /* one dot per horizontal line */
64 #define OVFLAG 0x20
65 #define SIGNFLAG 0x40
66 #define BLITREVERSE 0x2
67
68 #define SUD 0x10
69 #define SUL 0x8
70 #define AUL 0x4
71
72 #define OCTANT8 24
73 #define OCTANT7 4
74 #define OCTANT6 12
75 #define OCTANT5 28
76 #define OCTANT4 20
77 #define OCTANT3 8
78 #define OCTANT2 0
79 #define OCTANT1 16
80
81 /* stuff for blit geuer */
82 struct bltnode
83 {
84 struct bltnode *n;
85 int (*function)();
86 char stat;
87 short blitsize;
88 short beamsync;
89 int (*cleanup)();
90 };
91
92 /* defined bits for bltstat */
93 #define CLEANUP 0x40
94 #define CLEANME CLEANUP
95
96 #endif !HARDWARE_BLIT_H

```

```

1 /*****
2 /* Commodore-Amiga, Inc. */
3 /* cia.h */
4 /*****
5
6 #define CIAANAME "ciaa.resource"
7 #define CIABNAME "ciab.resource"
8

```

```

1 /*****
2 * Commodore-Amiga, Inc.
3 * custom.h
4 *
5 * $Header: custom.h,v 27.1 85/06/24 14:42:53 neil Exp $
6 *
7 * $Locker: $
8 *
9 *****/
10
11 #ifndef HARDWARE_CUSTOM_H
12 #define HARDWARE_CUSTOM_H
13
14 /*
15 * do this to get base of custom registers:
16 * extern struct Custom custom;
17 */
18
19
20 struct Custom {
21 UWORD bltddat;
22 UWORD dmaconr;
23 UWORD vposr;
24 UWORD vhposr;
25 UWORD dskdatr;
26 UWORD joy0dat;
27 UWORD joyldat;
28 UWORD clxdat;
29 UWORD adkconr;
30 UWORD pot0dat;
31 UWORD potldat;
32 UWORD potinp;
33 UWORD serdatr;
34 UWORD dskbytr;
35 UWORD intenar;
36 UWORD intregr;
37 APTR dskptr;
38 UWORD dsklen;
39 UWORD dskdat;
40 UWORD refptr;
41 UWORD vposw;
42 UWORD vhposw;
43 UWORD copcon;
44 UWORD serdat;
45 UWORD serper;
46 UWORD potgo;
47 UWORD joytest;
48 UWORD strequ;
49 UWORD strvbl;
50 UWORD strhor;
51 UWORD strlong;
52 UWORD bltcon0;
53 UWORD bltconl;
54 UWORD bltafwm;
55 UWORD bltalwm;
56 APTR bltcpt;
57 APTR bltbpt;
58 APTR bltapt;
59 APTR bltdpt;

```

```

60 UWORD bltsize;
61 UWORD pad2d[3];
62 UWORD bltcmmod;
63 UWORD bltbmod;
64 UWORD bltamod;
65 UWORD bltdmod;
66 UWORD pad34[4];
67 UWORD bltcdat;
68 UWORD bltbdat;
69 UWORD bltadat;
70 UWORD pad3b[4];
71 UWORD dsksync;
72 ULONG copllc;
73 ULONG cop2lc;
74 UWORD copjmpl;
75 UWORD copjmp2;
76 UWORD copins;
77 UWORD diwstrt;
78 UWORD diwstop;
79 UWORD ddfstrt;
80 UWORD ddfstop;
81 UWORD dmacon;
82 UWORD clxcon;
83 UWORD intena;
84 UWORD intreg;
85 UWORD adkcon;
86 struct AudChannel {
87 UWORD *ac_ptr; /* ptr to start of waveform data */
88 UWORD ac_len; /* length of waveform in words */
89 UWORD ac_per; /* sample period */
90 UWORD ac_vol; /* volume */
91 UWORD ac_dat; /* sample pair */
92 UWORD ac_pad[2]; /* unused */
93 } aud[4];
94 APTR bplpt[6];
95 UWORD pad7c[4];
96 UWORD bplcon0;
97 UWORD bplcon1;
98 UWORD bplcon2;
99 UWORD pad83;
100 UWORD bpl1mod;
101 UWORD bpl2mod;
102 UWORD pad86[2];
103 UWORD bpldat[6];
104 UWORD pad8e[2];
105 APTR sprpt[8];
106 struct SpriteDef {
107 UWORD pos;
108 UWORD ctl;
109 UWORD dataa;
110 UWORD datab;
111 } spr[8];
112 UWORD color[32];
113 };
114 #endif !HARDWARE_CUSTOM_H

```

```

1 /*****
2 * Commodore-Amiga, Inc.
3 * dmabits.h
4 *
5 * $Header: dmabits.h,v 27.1 85/06/24 14:42:59 neil Exp $
6 *
7 * $Locker: $
8 *
9 *****/
10
11 #ifndef HARDWARE_DMABITS_H
12 #define HARDWARE_DMABITS_H
13
14 /* include file for defining dma control stuff */
15
16 /* write definitions for dmaconw */
17 #define DMAF_SETCLR 0x8000
18 #define DMAF_AUDIO 0x000F /* 4 bit mask */
19 #define DMAF_AUD0 0x0001
20 #define DMAF_AUD1 0x0002
21 #define DMAF_AUD2 0x0004
22 #define DMAF_AUD3 0x0008
23 #define DMAF_DISK 0x0010
24 #define DMAF_SPRITE 0x0020
25 #define DMAF_BLITTER 0x0040
26 #define DMAF_COPPER 0x0080
27 #define DMAF_RASTER 0x0100
28 #define DMAF_MASTER 0x0200
29 #define DMAF_BLITHOG 0x0400
30 #define DMAF_ALL 0x01FF /* all dma channels */
31
32 /* read definitions for dmaconr */
33 /* bits 0-8 correspnd to dmaconw definitions */
34 #define DMAF_BLTDONE 0x4000
35 #define DMAF_BLTNZERO 0x2000
36
37 #define DMAB_SETCLR 15
38 #define DMAB_AUD0 0
39 #define DMAB_AUD1 1
40 #define DMAB_AUD2 2
41 #define DMAB_AUD3 3
42 #define DMAB_DISK 4
43 #define DMAB_SPRITE 5
44 #define DMAB_BLITTER 6
45 #define DMAB COPPER 7
46 #define DMAB_RASTER 8
47 #define DMAB_MASTER 9
48 #define DMAB BLITHOG 10
49 #define DMAB BLTDONE 14
50 #define DMAB BLTNZERO 13
51
52
53 #endif !HARDWARE_DMABITS_H

```



```

1 /*****
2 * Commodore-Amiga, Inc.
3 * intenabits.h -- definitions for the bits in the interrupt enable
4 * (and interrupt request) register
5 *
6 * $Header: intbits.h,v 27.1 85/06/24 14:43:04 neil Exp $
7 *
8 * $Locker: $
9 *
10 *****/
11
12 #ifndef HARDWARE_INTBITS_H
13 #define HARDWARE_INTBITS_H
14
15 #define INTB_SETCLR (15) /* Set/Clear control bit. Determines if bits */
16 /* written with a 1 get set or cleared. Bits */
17 /* written with a zero are always unchanged */
18 #define INTB_INTEN (14) /* Master interrupt (enable only) */
19 #define INTB_EXTER (13) /* External interrupt */
20 #define INTB_DSKSYNC (12) /* Disk re-SYNChronized */
21 #define INTB_RBF (11) /* serial port Receive Buffer Full */
22 #define INTB_AUD3 (10) /* Audio channel 3 block finished */
23 #define INTB_AUD2 (9) /* Audio channel 2 block finished */
24 #define INTB_AUD1 (8) /* Audio channel 1 block finished */
25 #define INTB_AUD0 (7) /* Audio channel 0 block finished */
26 #define INTB_BLIT (6) /* Blitter finished */
27 #define INTB_VERTB (5) /* start of Vertical Blank */
28 #define INTB_COPER (4) /* Coprocessor */
29 #define INTB_PORTS (3) /* I/O Ports and timers */
30 #define INTB_SOFTINT (2) /* software interrupt request */
31 #define INTB_DSKBLK (1) /* Disk Block done */
32 #define INTB_TBE (0) /* serial port Transmit Buffer Empty */
33
34
35
36 #define INTF_SETCLR (1<<15)
37 #define INTF_INTEN (1<<14)
38 #define INTF_EXTER (1<<13)
39 #define INTF_DSKSYNC (1<<12)
40 #define INTF_RBF (1<<11)
41 #define INTF_AUD3 (1<<10)
42 #define INTF_AUD2 (1<<9)
43 #define INTF_AUD1 (1<<8)
44 #define INTF_AUD0 (1<<7)
45 #define INTF_BLIT (1<<6)
46 #define INTF_VERTB (1<<5)
47 #define INTF_COPER (1<<4)
48 #define INTF_PORTS (1<<3)
49 #define INTF_SOFTINT (1<<2)
50 #define INTF_DSKBLK (1<<1)
51 #define INTF_TBE (1<<0)
52
53 #endif !HARDWARE_INTBITS_H

```

## Contents

```

intuition/intuinternal.h
intuition/intuition.h
intuition/intuitionbase.h

```

```

1: #ifndef INTUITION_INTUITION_H
2: #define INTUITION_INTUITION_H TRUE
3:
4: /** intuition.h *****/
5: * Commodore-Amiga, Inc.
6: *
7: * intuition.h main include for c programmers
8: *
9: * Modification History
10: * date : author : Comments
11: * -----
12: * 1-30-85 --RJ-- created this file!
13: * 10-03-85 Support for HP printers
14: *
15: *****/
16:
17: #ifndef INTUITION_INTUITIONBASE_H
18: #include "intuition/intuitionbase.h"
19: #endif
20:
21: #ifndef GRAPHICS_GFX_H
22: #include "graphics/gfx.h"
23: #endif
24:
25: #ifndef GRAPHICS_CLIP_H
26: #include "graphics/clip.h"
27: #endif
28:
29: #ifndef GRAPHICS_VIEW_H
30: #include "graphics/view.h"
31: #endif
32:
33: #ifndef GRAPHICS_RASTPORT_H
34: #include "graphics/rastport.h"
35: #endif
36:
37: #ifndef GRAPHICS_LAYERS_H
38: #include "graphics/layers.h"
39: #endif
40:
41: #ifndef GRAPHICS_TEXT_H
42: #include "graphics/text.h"
43: #endif
44:
45: #ifndef EXEC_PORTS_H
46: #include "exec/ports.h"
47: #endif
48:
49: #ifndef DEVICES_TIMER_H
50: #include "devices/timer.h"
51: #endif
52:
53: #ifndef DEVICES_INPTEVENT_H
54: #include "devices/inpthevent.h"
55: #endif
56:
57:
58: /* =====
59: /* == Menu =====
60: /* =====
61: struct Menu
62: {
63: struct Menu *NextMenu; /* same level */
64: SHORT LeftEdge, TopEdge; /* position of the select
65: box */
66: SHORT Width, Height; /* dimensions of the
67: select box */
68: USHORT Flags; /* see flag definitions
69: below */
70: BYTE *MenuName; /* text for this Menu
71: Header */
72: struct MenuItem *FirstItem; /* pointer to first
73: in chain */
74:
75: /* these mysteriously-named variables are for internal
76: use only */
77: SHORT JazzX, JazzY, BeatX, BeatY;
78: };
79:
80: /* FLAGS SET BY BOTH THE APPLIPROG AND INTUITION */
81: #define MENUENABLED 0x0001 /* whether or not this
82: menu is enabled */
83:
84: /* FLAGS SET BY INTUITION */
85: #define MIDDRAWN 0x0100 /* this menu's items are
86: currently drawn */
87:
88: /* =====
89: /* == MenuItem =====
90: /* =====
91: struct MenuItem
92: {
93: struct MenuItem *NextItem; /* pointer to next in
94: chained list */
95: SHORT LeftEdge, TopEdge; /* position of the select
96: box */
97: SHORT Width, Height; /* dimensions of the
98: select box */
99: USHORT Flags; /* see the defines below
100: */
101: LONG MutualExclude; /* set bits mean this
102: item
103: excludes that */

```

```

98: APTR ItemFill; /* points to Image,
 IntuiText, or NULL */
99:
100: /* when this item is pointed to by the cursor and the
 items highlight
101: * mode HIGHIMAGE is selected, this alternate image
 will be displayed
102: */
103: APTR SelectFill; /* points to Image, IntuiText,
 or NULL */
104:
105: BYTE Command; /* only if appliprogram sets
 the COMMSEQ flag */
106:
107: struct MenuItem *SubItem; /* if non-zero, DrawMenu
 shows "-->" */
108:
109: /* The NextSelect field represents the menu number of
 next selected
110: * item (when user has drag-selected several items)
111: */
112: USHORT NextSelect;
113: };
114:
115:
116: /* FLAGS SET BY THE APPLIPROG */
117: #define CHECKIT 0x0001 /* whether to check this item
 if selected */
118: #define ITEMTEXT 0x0002 /* set if textual, clear if
 graphical item */
119: #define COMMSEQ 0x0004 /* set if there's an command
 sequence */
120: #define MENUTOGGLE 0x0008 /* set to toggle the check
 of a menu item */
121: #define ITEMENABLED 0x0010 /* set if this item is enabled
 */
122:
123: /* these are the SPECIAL HIGHLIGHT FLAG state meanings */
124: #define HIGHFLAGS 0x00C0 /* see definitions below for
 these bits */
125: #define HIGHIMAGE 0x0000 /* use the user's "select
 image" */
126: #define HIGHCOMP 0x0040 /* highlight by complementing
 the selectbox */
127: #define HIGHBOX 0x0080 /* highlight by "boxing" the
 selectbox */
128: #define HIGHNONE 0x00C0 /* don't highlight */
129:
130: /* FLAGS SET BY BOTH APPLIPROG AND INTUITION */
131: #define CHECKED 0x0100 /* if CHECKIT, then set this
 when selected */
132:
133: /* FLAGS SET BY INTUITION */
134: #define ISDRAWN 0x1000 /* this item's subs are currently
 drawn */
135: #define HIGHLIGHT 0x2000 /* this item is currently
 highlighted */
136: #define MENUTOGGLED 0x4000 /* this item was already toggled
 */

137:
138:
139:
140:
141:
142: /* =====
 */
143: /* == Requester ==
 */
144: /* =====
 */
145: struct Requester
146: {
147: /* the ClipRect and BitMap and used for rendering the
 requester */
148: struct Requester *OlderRequest;
149: SHORT LeftEdge, TopEdge; /* dimensions of the
 entire box */
150: SHORT Width, Height; /* dimensions of the
 entire box */
151: SHORT RelLeft, RelTop; /* for Pointer relativity
 offsets */
152:
153: struct Gadget *ReqGadget; /* pointer to a list
 of Gadgets */
154: struct Border *ReqBorder; /* the box's border
 */
155: struct IntuiText *ReqText; /* the box's text */
156: USHORT Flags; /* see definitions below
 */
157:
158: /* pen number for back-plane fill before draws */
159: UBYTE BackFill;
160: /* Layer in place of clip rect */
161: struct Layer *ReqLayer;
162:
163: UBYTE ReqPad1[32];
164:
165: /* If the BitMap plane pointers are non-zero, this tells
 the system
166: * that the image comes pre-drawn (if the appliprogram
 wants to define
167: * it's own box, in any shape or size it wants!); this
 is OK by
168: * Intuition as long as there's a good correspondence
 between
169: * the image and the specified Gadgets
170: */
171: struct BitMap *ImageBMap; /* points to the BitMap of
 PREDRAWN imagery */
172: struct Window *RWindow; /* added. points back to
 Window */
173: UBYTE ReqPad2[36]
174: };
175:
176:
177: /* FLAGS SET BY THE APPLIPROG */
178: #define POINTREL 0x0001 /* if POINTREL set, TopLeft is
 relative to pointer*/

```

```

179: #define PREDRAWN 0x0002 /* if ReqBMap points to predrawn
 Requester
180: imagery */
181: /* FLAGS SET BY BOTH THE APPLIPROG AND INTUITION */
182:
183: /* FLAGS SET BY INTUITION */
184: #define REQOFFWINDOW 0x1000 /* part of one of the Gadgets
 was offwindow */
185: #define REQACTIVE 0x2000 /* this requester is active
 */
186: #define SYSREQUEST 0x4000 /* this requester caused by
 system */
187: #define DEFERREFRESH 0x8000 /* this Requester stops a
 Refresh broadcast */
188:
189:
190:
191:
192:
193:
194: /* =====
 */
195: /* --- Gadget =====
 */
196: /* =====
 */
197: struct Gadget
198: [
199: struct Gadget *NextGadget; /* next gadget in the list
 */
200:
201: SHORT LeftEdge, TopEdge; /* "hit box" of gadget
 */
202: SHORT Width, Height; /* "hit box" of gadget
 */
203:
204: USHORT Flags; /* see below for list of
 defines */
205:
206: USHORT Activation; /* see below for list of
 defines */
207:
208: USHORT GadgetType; /* see below for defines
 */
209:
210: /* appliprogram can specify that the Gadget be rendered
 as either as Border
211: * or an Image. This variable points to which (or equals
 NULL if there's
212: * nothing to be rendered about this Gadget)
213: */
214: APTR GadgetRender;
215:
216: /* appliprogram can specify "highlighted" imagery rather
 than algorithmic
217: * this can point to either Border or Image data
218: */
219: APTR SelectRender;
220:
221: struct IntuiText *GadgetText; /* text for this gadget
 */
222:
223: /* by using the MutualExclude word, the appliprogram can
 describe
224: * which gadgets mutually-exclude which other ones.
 The bits
225: * in MutualExclude correspond to the gadgets in object
 containing
226: * the gadget list. If this gadget is selected and
 a bit is set
227: * in this gadget's MutualExclude and the gadget corresponding
 to
228: * that bit is currently selected (e.g. bit 2 set and
 gadget 2
229: * is currently selected) that gadget must be unselected.
230:
231: * Intuition does the visual unselecting (with checkmarks)
 and
232: * leaves it up to the program to unselect internally
233: */
234: LONG MutualExclude; /* set bits mean this gadget
 excludes
235:
236: /* pointer to a structure of special data required by
 Proportional,
237: * String and Integer Gadgets
238: */
239: APTR SpecialInfo;
240:
241: USHORT GadgetID; /* user-definable ID field
 */
242: APTR UserData; /* ptr to general purpose
 User data
243:
244: (ignored by In) */
245:];
246:
247: /* --- FLAGS SET BY THE APPLIPROG ---
 */
248: /* combinations in these bits describe the highlight technique
 to be used */
249: #define GADGHIGHBITS 0x0003
250: #define GADGHCOMP 0x0000 /* Complement the select
 box */
251: #define GADGHBOX 0x0001 /* Draw a box around the
 image */
252: #define GADGHIMAGE 0x0002 /* Blast in this alternate
 image */
253: #define GADGHNONE 0x0003 /* don't highlight */
254:
255: /* set this flag if the GadgetRender and SelectRender point
 to Image imagery,
256: * clear if it's a Border
257: */
258: #define GADGIMAGE 0x0004
259:
260: /* combinations in these next two bits specify to which
 corner the gadget's
261: * Left & Top coordinates are relative. If relative to

```

```

262: Top/Left,
263: * these are "normal" coordinates (everything is relative
264: * to something in
265: * this universe)
266: */
267: #define GRELBOTTOM 0x0008 /* set if rel to bottom,
 clear if rel top */
268: #define GRELRIGHT 0x0010 /* set if rel to right,
 clear if to left */
269: /* set the RELWIDTH bit to spec that Width is relative to
 width of screen */
270: #define GRELWIDTH 0x0020
271: /* set the RELHEIGHT bit to spec that Height is rel to height
 of screen */
272: #define GRELHEIGHT 0x0040
273: /* the SELECTED flag is initialized by you and set by Intuition.
 It
274: * specifies whether or not this Gadget is currently selected/highlighted
275: */
276: #define SELECTED 0x0080
277:
278: /* the GADGDISABLED flag is initialized by you and later
 set by Intuition
279: * according to your calls to On/OffGadget(). It specifies
 whether or not
280: * this Gadget is currently disabled from being selected
281: */
282: #define GADGDISABLED 0x0100
283:
284:
285: /* --- These are the Activation flag bits -----
 */
286: /* RELVERIFY is set if you want to verify that the pointer
 was still over
287: * the gadget when the select button was released
288: */
289: #define RELVERIFY 0x0001
290:
291: /* the flag GADGIMMEDIATE, when set, informs the caller
 that the gadget
292: * was activated when it was activated. this flag works
 in conjunction with
293: * the RELVERIFY flag
294: */
295: #define GADGIMMEDIATE 0x0002
296:
297: /* the flag ENDGADGET, when set, tells the system that this
 gadget, when
298: * selected, causes the Requester or AbsMessage to be ended.
 Requesters or
299: * AbsMessages that are ended are erased and unlinked from
 the system */
300: #define ENDGADGET 0x0004
301:
302: /* the FOLLOWMOUSE flag, when set, specifies that you want
 to receive
303: * reports on mouse movements (ie, you want the REPORTMOUSE
 function for
304: * your Window). When the Gadget is deselected (immediately
 if you have
305: * no RELVERIFY) the previous state of the REPORTMOUSE flag
 is restored
306: * you probably want to set the GADGIMMEDIATE flag when
 using FOLLOWMOUSE,
307: * since that's the only reasonable way you have of learning
 why Intuition
308: * is suddenly sending you a stream of mouse movement events.
 If you don't
309: * set RELVERIFY, you'll get at least one Mouse Position
 event.
310: */
311: #define FOLLOWMOUSE 0x0008
312:
313: /* if any of the BORDER flags are set in a Gadget that's
 included in the
314: * Gadget list when a Window is opened, the corresponding
 Border will
315: * be adjusted to make room for the Gadget
316: */
317: #define RIGHTBORDER 0x0010
318: #define LEFTBORDER 0x0020
319: #define TOPBORDER 0x0040
320: #define BOTTOMBORDER 0x0080
321:
322: #define TOGGLESELECT 0x0100 /* this bit for toggle-select
 mode */
323:
324: #define STRINGCENTER 0x0200 /* should be a StringInfo
 flag, but it's OK*/
325: #define STRINGRIGHT 0x0400 /* should be a StringInfo
 flag, but it's OK*/
326:
327: #define LONGINT 0x0800 /* this String Gadget is
 actually LONG Int */
328:
329: #define ALTKEYMAP 0x1000 /* this String has an alternate
 keymap */
330:
331:
332: /* --- GADGET TYPES -----
 */
333: /* These are the Gadget Type definitions for the variable
 GadgetType
334: * gadget number type MUST start from one. NO TYPES OF
 ZERO ALLOWED.
335: * first comes the mask for Gadget flags reserved for Gadget
 typing
336: */
337: #define GADGETTYPE 0xFC00 /* all Gadget Global Type
 flags (padded) */
338: #define SYSGADGET 0x8000 /* 1 = SysGadget, 0 = AppliGadget
 */
339: #define SCRGADGET 0x4000 /* 1 = ScreenGadget, 0
 = WindowGadget */
340: #define GZZGADGET 0x2000 /* 1 = Gadget for GIMMEZEROZERO
 borders */
341: #define REQGADGET 0x1000 /* 1 = this is a Requester
 Gadget */

```

```

342: /* system gadgets */
343: #define SIZING 0x0010
344: #define WDRAGGING 0x0020
345: #define SDRAGGING 0x0030
346: #define WUPFRONT 0x0040
347: #define SUPFRONT 0x0050
348: #define WDOWNBACK 0x0060
349: #define SDOWNBACK 0x0070
350: #define CLOSE 0x0080
351: /* application gadgets */
352: #define BOOLGADGET 0x0001
353: #define GADGET0002 0x0002
354: #define PROPGADGET 0x0003
355: #define STRGADGET 0x0004
356:
357:
358:
359:
360:
361:
362: /* =====
 */
363: /* == PropInfo =====
 */
364: /* =====
 */
365: /* this is the special data required by the proportional
 Gadget
366: * typically, this data will be pointed to by the Gadget
 variable SpecialInfo
367: */
368: struct PropInfo
369: {
370: USHORT Flags; /* general purpose flag bits (see
 defines below) */
371:
372: /* You initialize the Pot variables before the Gadget
 is added to
373: * the system. Then you can look here for the current
 settings
374: * any time, even while User is playing with this Gadget.
 To
375: * adjust these after the Gadget is added to the System,
 use
376: * ModifyProp(); The Pots are the actual proportional
 settings,
377: * where a value of zero means zero and a value of MAXPOT
 means
378: * that the Gadget is set to its maximum setting.
379: */
380: USHORT HorizPot; /* 16-bit FixedPoint horizontal
 quantity percentage */
381: USHORT VertPot; /* 16-bit FixedPoint vertical quantity
 percentage */
382:
383: /* the 16-bit FixedPoint Body variables describe what
 percentage of
384: * the entire body of stuff referred to by this Gadget
 is actually
385: * shown at one time. This is used with the AUTOKNOB

```

```

routines,
386: * to adjust the size of the AUTOKNOB according to how
 much of
387: * the data can be seen. This is also used to decide
 how far
388: * to advance the Pots when User hits the Container
 of the Gadget.
389: * For instance, if you were controlling the display
 of a 5-line
390: * Window of text with this Gadget, and there was a
 total of 15
391: * lines that could be displayed, you would set the
 VertBody value to
392: * (MAXBODY / (TotalLines / DisplayLines)) = MAXBODY
 / 3.
393: * Therefore, the AUTOKNOB would fill 1/3 of the container,
 and
394: * if User hits the Container outside of the knob, the
 pot would
395: * advance 1/3 (plus or minus) If there's no body to
 show, or
396: * the total amount of displayable info is less than
 the display area,
397: * set the Body variables to the MAX. To adjust these
 after the
398: * Gadget is added to the System, use ModifyProp();
399: */
400: USHORT HorizBody; /* horizontal Body */
401: USHORT VertBody; /* vertical Body */
402:
403: /* these are the variables that Intuition sets and maintains
 */
404: USHORT CWidth; /* Container width (with any relativity
 absolved) */
405: USHORT CHeight; /* Container height (with any relativity
 absolved) */
406: USHORT HPotRes, VPotRes; /* pot increments */
407: USHORT LeftBorder; /* Container borders */
408: USHORT TopBorder; /* Container borders */
409: };
410:
411:
412: /* --- FLAG BITS ---
 */
413: #define AUTOKNOB 0x0001 /* this flag sez: gimme
 that old auto-knob*/
414: #define FREEHORIZ 0x0002 /* if set, the knob can
 move horizontally */
415: #define FREEVERT 0x0004 /* if set, the knob can
 move vertically */
416: #define PROPBORDERLESS 0x0008 /* if set, no border will
 be rendered */
417: #define KNOBHIT 0x0100 /* set when this Knob is
 hit */
418:
419: #define KNOBHMN 6 /* minimum horizontal size
 of the Knob */
420: #define KNOBVMIN 4 /* minimum vertical size
 of the Knob */

```

```

421: #define MAXBODY 0xFFFF /* maximum body value */
422: #define MAXPOT 0xFFFF /* maximum pot value */
423:
424:
425:
426:
427:
428:
429: /* =====
 */
430: /* --- StringInfo ---
 */
431: /* =====
 */
432: /* this is the special data required by the string Gadget
433: * typically, this data will be pointed to by the Gadget
 variable SpecialInfo
434: */
435: struct StringInfo
436: {
437: /* you initialize these variables, and then Intuition
 maintains them */
438: UBYTE *Buffer; /* the buffer containing the start
 and final string */
439: UBYTE *UndoBuffer; /* optional buffer for undoing
 current entry */
440: SHORT BufferPos; /* character position in Buffer
 */
441: SHORT MaxChars; /* max number of chars in Buffer
 (including NULL) */
442: SHORT DispPos; /* Buffer position of first displayed
 character */
443:
444: /* Intuition initializes and maintains these variables
 for you */
445: SHORT UndoPos; /* character position in the undo
 buffer */
446: SHORT NumChars; /* number of characters currently
 in Buffer */
447: SHORT DispCount; /* number of whole characters visible
 in Container */
448: SHORT CLeft, CTop; /* topleft offset of the container
 */
449: struct Layer *LayerPtr; /* the RastPort containing
 this Gadget */
450:
451: /* you can initialize this variable before the gadget
 is submitted to
452: * Intuition, and then examine it later to discover
 what integer
453: * the user has entered (if the user never plays with
 the gadget,
454: * the value will be unchanged from your initial setting)
455: */
456: LONG LongInt;
457:
458: /* If you want this Gadget to use your own Console keymapping,
 you
459: * set the ALTKEYMAP bit in the Activation flags of
 the Gadget, and then
460: * set this variable to point to your keymap. If you
 don't set the
461: * ALTKEYMAP, you'll get the standard ASCII keymapping.
462: */
463: struct KeyMap *AltKeyMap;
464: };
465:
466:
467:
468:
469:
470:
471: /* =====
 */
472: /* --- IntuiText ---
 */
473: /* =====
 */
474: /* IntuiText is a series of strings that start with a screen
 location
475: * (always relative to the upper-left corner of something)
 and then the
476: * text of the string. The text is null-terminated.
477: */
478: struct IntuiText
479: {
480: UBYTE FrontPen, BackPen; /* the pen numbers for
 the rendering */
481: UBYTE DrawMode; /* the mode for rendering
 the text */
482: SHORT LeftEdge; /* relative start location
 for the text */
483: SHORT TopEdge; /* relative start location
 for the text */
484: struct TextAttr *ITextFont; /* if NULL, you accept
 the default */
485: UBYTE *IText; /* pointer to null-terminated
 text */
486: struct IntuiText *NextText; /* continuation to TxWrite
 another text */
487: };
488:
489:
490:
491:
492:
493:
494: /* =====
 */
495: /* --- Border ---
 */
496: /* =====
 */
497: /* Data type Border, used for drawing a series of lines
 which is intended for
498: * use as a border drawing, but which may, in fact, be
 used to render any
499: * arbitrary vector shape.
500: * The routine DrawBorder sets up the RastPort with the
 appropriate

```

```

501: * variables, then does a Move to the first coordinate,
 then does Draws
502: * to the subsequent coordinates.
503: * After all the Draws are done, if NextBorder is non-zero
 we call DrawBorder
504: * recursively
505: */
506: struct Border
507: {
508: SHORT LeftEdge, TopEdge; /* initial offsets from
 the origin */
509: UBYTE FrontPen, BackPen; /* pens numbers for
 rendering */
510: UBYTE DrawMode; /* mode for rendering
 */
511: BYTE Count; /* number of XY pairs
 */
512: SHORT *XY; /* vector coordinate
 pairs relative
513: to LeftTop */
514: struct Border *NextBorder; /* pointer to any other
 Border too */
515: };
516:
517:
518:
519:
520:
521:
522: /* =====
 */
523: /* --- Image =====
 */
524: /* =====
 */
525: /* This is a brief image structure for very simple transfers
 of
526: * image data to a RastPort
527: */
528: struct Image
529: {
530: SHORT LeftEdge; /* starting offset relative
 to some origin */
531: SHORT TopEdge; /* starting offsets relative
 to some origin*/
532: SHORT Width; /* pixel size (though data
 is word-aligned)*/
533: SHORT Height, Depth; /* pixel sizes */
534: USHORT *ImageData; /* pointer to the actual
 word-aligned bits */
535:
536: /* the PlanePick and PlaneOnOff variables work much
 the same way as the
537: * equivalent GELS Bob variables. It's a space-saving
538: * mechanism for image data. Rather than defining the
 image data
539: * for every plane of the RastPort, you need define
 data only
540: * for the planes that are not entirely zero or one.
541:
542: * define your Imagery, you will often find that most
 of the planes
543: * ARE just as color selectors. For instance, if you're
 designing
544: * a two-color Gadget to use colors two and three, and
 the Gadget
545: * will reside in a five-plane display, bit plane zero
 of your
546: * imagery would be all ones, bit plane one would have
 data that
547: * describes the imagery, and bit planes two through
 four would be
548: * all zeroes. Using these flags allows you to avoid
 wasting all
549: * that memory in this way: first, you specify which
 planes you
550: * want your data to appear in using the PlanePick variable.
 For
551: * each bit set in the variable, the next "plane" of
 your image
552: * data is blitted to the display. For each bit clear
 in this
553: * variable, the corresponding bit in PlaneOnOff is
 examined.
554: * If that bit is clear, a "plane" of zeroes will be
 used.
555: * If the bit is set, ones will go out instead. So,
 for our example:
556: * Gadget.PlanePick = 0x02;
557: * Gadget.PlaneOnOff = 0x01;
558: * Note that this also allows for generic Gadgets, like
 the
559: * System Gadgets, which will work in any number of
 bit planes.
560: * Note also that if you want an Image that is only
 a filled
561: * rectangle, you can get this by setting PlanePick
 to zero
562: * (pick no planes of data) and set PlaneOnOff to describe
 the pen
563: * color of the rectangle.
564: */
565: UBYTE PlanePick, PlaneOnOff;
566:
567: /* if the NextImage variable is not NULL, Intuition
 presumes that
568: * it points to another Image structure with another
 Image to be
569: * rendered
570: */
571: struct Image *NextImage;
572: };
573:
574:
575:
576:
577:
578: /* =====

```



```

579: /*
580: /* -----
581: struct IntuiMessage
582: {
583: struct Message ExecMessage;
584:
585: /* the Class bits correspond directly with the IDCMP
586: Flags, except for the
587: * special bit LONELYMESSAGE (defined below)
588: */
589: ULONG Class;
590:
591: /* the Code field is for special values like MENU number
592: */
593: USHORT Code;
594:
595: /* the Qualifier field is a copy of the current InputEvent's
596: Qualifier */
597: USHORT Qualifier;
598:
599: /* IAddress contains particular addresses for Intuition
600: functions, like
601: * the pointer to the Gadget or the Screen
602: */
603: APTR IAddress;
604:
605: /* when getting mouse movement reports, any event you
606: get will have the
607: * the mouse coordinates in these variables. the coordinates
608: are relative
609: * to the upper-left corner of your Window (GIMMEZEROZERO
610: notwithstanding)
611: */
612: SHORT MouseX, MouseY;
613:
614: /* the time values are copies of the current system
615: clock time. Micros
616: * are in units of microseconds, Seconds in seconds.
617: */
618: ULONG Seconds, Micros;
619:
620: /* the IDCMPWindow variable will always have the address
621: of the Window of
622: * this IDCMP
623: */
624: struct Window *IDCMPWindow;
625:
626: /* system-use variable */
627: struct IntuiMessage *SpecialLink;
628: };
629:
630: /* --- IDCMP Classes -----
631: */
632: #define SIZEVERIFY 0x00000001 /* See the Programmer's
633: Guide */
634: #define NEWSIZE 0x00000002 /* See the Programmer's
635: Guide */
636:
637: #define REFRESHWINDOW 0x00000004 /* See the Programmer's
638: Guide */
639: #define MOUSEBUTTONS 0x00000008 /* See the Programmer's
640: Guide */
641: #define MOUSEMOVE 0x00000010 /* See the Programmer's
642: Guide */
643: #define GADGETDOWN 0x00000020 /* See the Programmer's
644: Guide */
645: #define GADGETUP 0x00000040 /* See the Programmer's
646: Guide */
647: #define REQSET 0x00000080 /* See the Programmer's
648: Guide */
649: #define MENUPICK 0x00000100 /* See the Programmer's
650: Guide */
651: #define CLOSEWINDOW 0x00000200 /* See the Programmer's
652: Guide */
653: #define RAWKEY 0x00000400 /* See the Programmer's
654: Guide */
655: #define REQVERIFY 0x00000800 /* See the Programmer's
656: Guide */
657: #define REQCLEAR 0x00001000 /* See the Programmer's
658: Guide */
659: #define MENUVERIFY 0x00002000 /* See the Programmer's
660: Guide */
661: #define NEWPREFS 0x00004000 /* See the Programmer's
662: Guide */
663: #define DISKINSERTED 0x00008000 /* See the Programmer's
664: Guide */
665: #define DISKREMOVED 0x00010000 /* See the Programmer's
666: Guide */
667: #define WBENCHMESSAGE 0x00020000 /* See the Programmer's
668: Guide */
669: #define ACTIVEWINDOW 0x00040000 /* See the Programmer's
670: Guide */
671: #define INACTIVEWINDOW 0x00080000 /* See the Programmer's
672: Guide */
673: #define DELTAMOVE 0x00100000 /* See the Programmer's
674: Guide */
675: #define VANILLAKEY 0x00200000 /* See below */
676: #define INTUITICKS 0x00400000 /* See below */
677: /* NOTEZ-BIEN: 0x80000000 is reserved for internal
678: use */
679:
680: #define VANILLAKEY
681: This is the raw keycode RAWKEY event translated into the
682: current
683: default character keymap of the Console Device. In the
684: USA, the
685: default keymap is ASCII character. When you set this flag,
686: you
687: will get IntuiMessages where the Code field has a character
688: representing the key struck on the keyboard. This character
689: is
690: from the default character KeyMap of the Console Device.
691:
692: */
693:
694: */
695:
696: */

```

```

659: INTUITICKS
660: You can get simple timer events from Intuition when your
 window is
661: the active one, which may help you avoid opening and managing
 the
662: timer device. Intuition receives timer events 10 times
 a
663: second (approximately). You can receive these events too,
 by setting
664: the INTUITICKS flag. You will only get one queued-up INTUITICKS
 message at a time; if Intuition notices that you've been
665: sent an
666: INTUITICKS message but you haven't replied to it yet, another
 message
667: will NOT be sent.
668: */
669:
670: /* the IDCMP Flags do not use this special bit, which is
 cleared when
671: * Intuition sends its special message to the Task, and
 set when Intuition
672: * gets its Message back from the Task. Therefore, I can
 check here to
673: * find out fast whether or not this Message is available
 for me to send.
674: */
675: #define LONELYMESSAGE 0x80000000
676:
677:
678: /* --- IDCMP Codes -----
 */
679: /* This group of codes is for the MENUVERIFY function */
680: #define MENUHOT 0x0001 /* IntuiWants verification
 or MENUCANCEL */
681: #define MENUCANCEL 0x0002 /* HOT Reply of this cancels
 Menu operation */
682: #define MENUWAITING 0x0003 /* Intuition simply wants
 a ReplyMsg() ASAP */
683:
684: /* This group of codes is for the WBENCHMESSAGE messages
 */
685: #define WBENCHOPEN 0x0001
686: #define WBENCHCLOSE 0x0002
687:
688:
689:
690: /* =====
 */
691: /* --- Window -----
 */
692: /* =====
 */
693: struct Window
694: {
695: struct Window *NextWindow; /* for the linked list
 in a screen */
696:
697: SHORT LeftEdge, TopEdge; /* screen dimensions
 of window */

```

```

569: SHORT Width, Height; /* screen dimensions
 of window */
569:
570: SHORT MouseY, MouseX; /* relative to upper-left
 of window */
571:
572: SHORT MinWidth, MinHeight; /* minimum sizes */
573: SHORT MaxWidth, MaxHeight; /* maximum sizes */
574:
575: ULONG Flags; /* see below for defines
 */
576:
577: struct Menu *MenuStrip; /* the strip of Menu
 headers */
578:
579: UBYTE *Title; /* the title text for
 this window */
580:
581: struct Requester *FirstRequest; /* all active Requesters
 */
582:
583: struct Requester *DMRequest; /* double-click Requester
 */
584:
585: SHORT ReqCount; /* count of reqs blocking
 Window */
586:
587: struct Screen *WScreen; /* this Window's Screen
 */
588: struct RastPort *RPort; /* this Window's very
 own RastPort */
589:
590: /* the border variables describe the window border.
 If you specify
591: * GIMMEZEROZERO when you open the window, then the
 upper-left of the
592: * ClipRect for this window will be upper-left of the
 BitMap (with correct
593: * offsets when in SuperBitMap mode; you MUST select
 GIMMEZEROZERO when
594: * using SuperBitMap). If you don't specify ZeroZero,
 then you save
595: * memory (no allocation of RastPort, Layer, ClipRect
 and associated
596: * Bitmaps), but you also must offset all your writes
 by BorderTop,
597: * BorderLeft and do your own mini-clipping to prevent
 writing over the
598: * system gadgets
599: */
600: BYTE BorderLeft, BorderTop, BorderRight, BorderBottom;
601: struct RastPort *BorderRPort;
602:
603: /* You supply a linked-list of Gadgets for your Window.
 * This list DOES NOT include system gadgets. You get
 the standard
604: * window system gadgets by setting flag-bits in the
 variable Flags (see
605: * the bit definitions below)

```

```

738: */
739: struct Gadget *FirstGadget;
740:
741: /* these are for opening/closing the windows */
742: struct Window *Parent, *Descendant;
743:
744: /* sprite data information for your own Pointer
745: * set these AFTER you Open the Window by calling SetPointer()
746: */
747: USHORT *Pointer; /* sprite data */
748: BYTE PtrHeight; /* sprite height (not
including
749: sprite padding) */
750: BYTE PtrWidth; /* sprite width (must
be less than or
751: equal to 16) */
752: BYTE XOffset, YOffset; /* sprite offsets */
753:
754: /* the IDCMP Flags and User's and Intuition's Message
Ports */
755: ULONG IDCMPFlags; /* User-selected flags
*/
756: struct MsgPort *UserPort, *WindowPort;
757: struct IntuiMessage *MessageKey;
758:
759: UBYTE DetailPen, BlockPen; /* for bar/border/gadget
rendering */
760:
761: /* the CheckMark is a pointer to the imagery that will
be used when
762: * rendering MenuItems of this Window that want to be
checkmarked
763: * if this is equal to NULL, you'll get the default
imagery
764: */
765: struct Image *CheckMark;
766:
767: UBYTE *ScreenTitle; /* if non-null, Screen title when
Window is active*/
768:
769: /* These variables have the mouse coordinates relative
to the
770: * inner-Window of GIMMEZEROZERO Windows. This is compared
with the
771: * MouseX and MouseY variables, which contain the mouse
coordinates
772: * relative to the upper-left corner of the Window,
GIMMEZEROZERO
773: * notwithstanding
774: */
775: SHORT GZZMouseX;
776: SHORT GZZMouseY;
777: /* these variables contain the width and height of the
inner-Window of
778: * GIMMEZEROZERO Windows
779: */
780: SHORT GZZWidth;
781: SHORT GZZHeight;
782:
783: UBYTE *ExtData;

```

```

784:
785: BYTE *UserData; /* general-purpose pointer to User
data extension */
786:
787: /** jimm: NEW: 11/18/85: this pointer keeps a duplicate
of what
788: * Window.RPort->Layer is _supposed_ to be pointing
at
789: */
790: struct Layer *WLayer;
791: };
792:
793:
794: /* --- FLAGS REQUESTED (NOT DIRECTLY SET THOUGH) BY THE
APPLIPROG ----- */
795: #define WINDOWSIZING 0x0001 /* include sizing system-gadget?
*/
796: #define WINDOWDRAG 0x0002 /* include dragging
system-gadget? */
797: #define WINDOWDEPTH 0x0004 /* include depth arrangement
gadget? */
798: #define WINDOWCLOSE 0x0008 /* include close-box
system-gadget? */
799:
800: #define SIZEBRIGHT 0x0010 /* size gadget uses
right border */
801: #define SIZEBOTTOM 0x0020 /* size gadget uses
bottom border */
802:
803: /* --- refresh modes -----
*/
804: /* combinations of the REFRESHBITS select the refresh type
*/
805: #define REFRESHBITS 0x00C0
806: #define SMART_REFRESH 0x0000
807: #define SIMPLE_REFRESH 0x0040
808: #define SUPER_BITMAP 0x0080
809: #define OTHER_REFRESH 0x00C0
810:
811: #define BACKDROP 0x0100 /* this is an ever-popular
BACKDROP window */
812:
813: #define REPORTMOUSE 0x0200 /* set this to hear about
every mouse move */
814:
815: #define GIMMEZEROZERO 0x0400 /* make extra border stuff
*/
816:
817: #define BORDERLESS 0x0800 /* set this to get a Window
sans border */
818:
819: #define ACTIVATE 0x1000 /* when Window opens, it's
the Active one */
820:
821: /* FLAGS SET BY INTUITION */
822: #define WINDOWACTIVE 0x2000 /* this window is the active
one */
823: #define INREQUEST 0x4000 /* this window is in request
mode */
824: #define MENUSTATE 0x8000 /* this Window is active

```

```

825: with its Menus on */
826: /* --- Other User Flags -----
*/
827: #define RMBTRAP 0x00010000 /* Catch RMB events
for your own */
828: #define NOCAREREFRESH 0x00020000 /* not to be bothered
with REFRESH */
829:
830:
831: /* --- Other Intuition Flags -----
*/
832: #define WINDOWREFRESH 0x01000000 /* Window is currently
refreshing */
833: #define WBENCHWINDOW 0x02000000 /* WorkBench tool ONLY
Window */
834: #define WINDOWTICKED 0x04000000 /* only one timer tick
at a time */
835:
836: #define SUPER_UNUSED 0xFCFC0000 /* bits of Flag unused
yet */
837:
838:
839: /* --- see struct IntuiMessage for the IDCMP Flag definitions
----- */
840:
841:
842:
843:
844: /* =====
*/
845: /* === NewWindow =====
*/
846: /* =====
*/
847: struct NewWindow
848: {
849: SHORT LeftEdge, TopEdge; /* screen dimensions
of window */
850: SHORT Width, Height; /* screen dimensions
of window */
851:
852: UBYTE DetailPen, BlockPen; /* for bar/border/gadget
rendering */
853:
854: ULONG IDCMPFlags; /* User-selected IDCMP
flags */
855:
856: ULONG Flags; /* see Window struct
for defines */
857:
858: /* You supply a linked-list of Gadgets for your Window.
859: * This list DOES NOT include system Gadgets. You
get the standard
860: * system Window Gadgets by setting flag-bits in the
variable Flags (see
861: * the bit definitions under the Window structure definition)
862: */
863: struct Gadget *FirstGadget;
864:
865: /* the CheckMark is a pointer to the imagery that will
be used when
866: * rendering MenuItems of this Window that want to be
checkmarked
867: * if this is equal to NULL, you'll get the default
imagery
868: */
869: struct Image *CheckMark;
870:
871: UBYTE *Title; /* the title text for
this window */
872:
873: /* the Screen pointer is used only if you've defined
a CUSTOMSCREEN and
874: * want this Window to open in it. If so, you pass
the address of the
875: * Custom Screen structure in this variable. Otherwise,
this variable
876: * is ignored and doesn't have to be initialized.
877: */
878: struct Screen *Screen;
879:
880: /* SUPER_BITMAP Window? If so, put the address of your
Bitmap structure
881: * in this variable. If not, this variable is ignored
and doesn't have
882: * to be initialized
883: */
884: struct Bitmap *Bitmap;
885:
886: /* the values describe the minimum and maximum sizes
of your Windows.
887: * these matter only if you've chosen the WINDOWSIZING
Gadget option,
888: * which means that you want to let the User to change
the size of
889: * this Window. You describe the minimum and maximum
sizes that the
890: * Window can grow by setting these variables. You
can initialize
891: * any one these to zero, which will mean that you want
to duplicate
892: * the setting for that dimension (if MinWidth == 0,
MinWidth will be
893: * set to the opening Width of the Window).
894: * You can change these settings later using SetWindowLimits().
895: * If you haven't asked for a SIZING Gadget, you don't
have to
896: * initialize any of these variables.
897: */
898: SHORT MinWidth, MinHeight; /* minimums */
899: SHORT MaxWidth, MaxHeight; /* maximums */
900:
901: /* the type variable describes the Screen in which you
want this Window to
902: * open. The type value can either be CUSTOMSCREEN
or one of the
903: * system standard Screen Types such as WBENCHSCREEN.
See the
904: * type definitions under the Screen structure

```

```

905: */
906: USHORT Type;
907: };
908:
909:
910:
911:
912:
913:
914: /* =====
915: /* == Screen =====
916: /* =====
917: struct Screen
918: {
919: struct Screen *NextScreen; /* linked list of screens
920: */
921: struct Window *FirstWindow; /* linked list Screen's
922: Windows */
923: SHORT LeftEdge, TopEdge; /* parameters of the
924: screen */
925: SHORT Width, Height; /* parameters of the
926: screen */
927: SHORT MouseY, MouseX; /* position relative
928: to upper-left */
929: USHORT Flags; /* see definitions below
930: */
931: UBYTE *Title; /* null-terminated Title
932: text */
933: UBYTE *DefaultTitle; /* for Windows without
934: ScreenTitle */
935:
936: /* Bar sizes for this Screen and all Window's in this
937: Screen */
938: BYTE BarHeight, BarVBorder, BarHBorder, MenuVBorder,
939: MenuHBorder;
940: BYTE WBotTop, WBotLeft, WBotRight, WBotBottom;
941:
942: struct TextAttr *Font; /* this screen's default
943: font */
944:
945: /* the display data structures for this Screen */
946: struct ViewPort ViewPort; /* describing the Screen's
947: display */
948: struct RastPort RastPort; /* describing Screen
949: rendering */
950: struct BitMap BitMap; /* auxiliary graphexcess
951: baggage */
952: struct Layer_Info LayerInfo; /* each screen gets
953: a LayerInfo */
954:
955: /* You supply a linked-list of Gadgets for your Screen.
956: * This list DOES NOT include system Gadgets. You
957: get the standard

```

```

946: * system Screen Gadgets by default
947: */
948: struct Gadget *FirstGadget;
949:
950: UBYTE DetailPen, BlockPen; /* for bar/border/gadget
951: rendering */
952:
953: /* the following variable(s) are maintained by Intuition
954: to support the
955: * DisplayBeep() color flashing technique
956: */
957: USHORT SaveColor0;
958:
959: /* This layer is for the Screen and Menu bars */
960: struct Layer *BarLayer;
961:
962: UBYTE *ExtData;
963:
964: UBYTE *UserData; /* general-purpose pointer to User
965: data extension */
966: };
967:
968: /* --- FLAGS SET BY INTUITION -----
969: */
970: /* The SCREENTYPE bits are reserved for describing various
971: Screen types
972: * available under Intuition.
973: */
974: #define SCREENTYPE 0x000F /* all the screens types
975: available */
976: /* --- the definitions for the Screen Type -----
977: */
978: #define WBENCHSCREEN 0x0001 /* Ta Da! The Workbench
979: */
980: #define CUSTOMSCREEN 0x000F /* for that special
981: look */
982:
983: #define SHOWTITLE 0x0010 /* this gets set by
984: a call to
985: ShowTitle()*/
986:
987: #define BEEPING 0x0020 /* set when Screen is
988: beeping */
989:
990: #define CUSTOMBITMAP 0x0040 /* if you are supplying
991: your own BitMap */
992:
993:
994:
995:
996:
997: /* =====
998: /* == NewScreen =====
999: /* =====
1000: */
1001: struct NewScreen
1002: {
1003: SHORT LeftEdge, TopEdge, Width, Height, Depth; /*

```

```

screen dimensions */
991:
992: UBYTE DetailPen, BlockPen; /* for bar/border/gadget
rendering */
993:
994: USHORT ViewModes; /* the Modes for the
ViewPort (and View)*/
995:
996: USHORT Type; /* the Screen type (see
defines above) */
997:
998: struct TextAttr *Font; /* this Screen's default
text attributes*/
999:
1000: UBYTE *DefaultTitle; /* the default title
for this Screen */
1001:
1002: struct Gadget *Gadgets; /* your own Gadgets
for this Screen */
1003:
1004: /* if you are opening a CUSTOMSCREEN and already have
a BitMap
1005: * that you want used for your Screen, you set the flags
CUSTOMBITMAP in
1006: * the Types variable and you set this variable to point
to your BitMap
1007: * structure. The structure will be copied into your
Screen structure,
1008: * after which you may discard your own BitMap if you
want
1009: */
1010: struct BitMap *CustomBitMap;
1011: };
1012:
1013:
1014:
1015:
1016: /* =====
*/
1017: /* == Preferences =====
*/
1018: /* =====
*/
1019:
1020: /* these are the definitions for the printer configurations
*/
1021: #define FILENAME_SIZE 30 /* Filename size */
1022:
1023: #define POINTERSIZE (1 + 16 + 1) * 2 /* Size of Pointer
data buffer */
1024:
1025: /* These defines are for the default font size. These actually
describe the
1026: * height of the defaults fonts. The default font type
is the topaz
1027: * font, which is a fixed width font that can be used in
either
1028: * eighty-column or sixty-column mode. The Preferences
structure reflects
1029: * which is currently selected by the value found in the

variable FontSize,
1030: * which may have either of the values defined below. These
values actually
1031: * are used to select the height of the default font. By
changing the
1032: * height, the resolution of the font changes as well.
1033: */
1034: #define TOPAZ_EIGHTY 8
1035: #define TOPAZ_SIXTY 9
1036:
1037:
1038: struct Preferences
1039: {
1040: /* the default font height */
1041: BYTE FontHeight; /* height for system
default font */
1042:
1043: /* constant describing what's hooked up to the port
*/
1044: UBYTE PrinterPort; /* printer port connection
*/
1045:
1046: /* the baud rate of the port */
1047: USHORT BaudRate; /* baud rate for the serial
port */
1048:
1049: /* various timing rates */
1050: struct timeval KeyRptSpeed; /* repeat speed for
keyboard */
1051: struct timeval KeyRptDelay; /* Delay before keys
repeat */
1052: struct timeval DoubleClick; /* Interval allowed
between clicks */
1053:
1054: /* Intuition Pointer data */
1055: USHORT PointerMatrix[POINTERSIZE]; /* Definition of
pointer sprite */
1056: BYTE XOffset; /* X-Offset for active
'bit' */
1057: BYTE YOffset; /* Y-Offset for active
'bit' */
1058: USHORT color17; /******
1059: USHORT color18; /* Colours for sprite
pointer */
1060: USHORT color19; /******
1061: USHORT PointerTicks; /* Sensitivity of the
pointer */
1062:
1063: /* Workbench Screen colors */
1064: USHORT color0; /******
1065: USHORT color1; /* Standard default
colours */
1066: USHORT color2; /* Used in the Workbench
*/
1067: USHORT color3; /******
1068:
1069: /* positioning data for the Intuition View */
1070: BYTE ViewXOffset; /* Offset for top lefthand
corner */
1071: BYTE ViewYOffset; /* X and Y dimensions

```

```

1072: WORD ViewInitX, ViewInitY; /* View initial offset
values */
1073:
1074: BOOL EnableCLI; /* CLI availability
switch */
1075:
1076: /* printer configurations */
1077: USHORT PrinterType; /* printer type
*/
1078: UBYTE PrinterFilename[FILENAME_SIZE]; /* file for
printer */
1079:
1080: /* print format and quality configurations */
1081: USHORT PrintPitch; /* print pitch
*/
1082: USHORT PrintQuality; /* print quality
*/
1083: USHORT PrintSpacing; /* number of lines per
inch */
1084: UWORD PrintLeftMargin; /* left margin in characters
*/
1085: UWORD PrintRightMargin; /* right margin in characters
*/
1086: USHORT PrintImage; /* positive or negative
*/
1087: USHORT PrintAspect; /* horizontal or vertical
*/
1088: USHORT PrintShade; /* b&w, half-tone, or
color */
1089: WORD PrintThreshold; /* darkness ctrl for
b/w dumps */
1090:
1091: /* print paper descriptors */
1092: USHORT PaperSize; /* paper size
*/
1093: UWORD PaperLength; /* paper length in number
of lines */
1094: USHORT PaperType; /* continuous or single
sheet */
1095:
1096: BYTE padding[50]; /* For further system
expansion */
1097: };
1098:
1099:
1100: /* PrinterPort */
1101: #define PARALLEL_PRINTER 0x00
1102: #define SERIAL_PRINTER 0x01
1103:
1104: /* BaudRate */
1105: #define BAUD_110 0x00
1106: #define BAUD_300 0x01
1107: #define BAUD_1200 0x02
1108: #define BAUD_2400 0x03
1109: #define BAUD_4800 0x04
1110: #define BAUD_9600 0x05
1111: #define BAUD_19200 0x06
1112: #define BAUD_MIDI 0x07
1113:
1114: /* PaperType */
1115: #define FANFOLD 0x00
1116: #define SINGLE 0x80
1117:
1118: /* PrintPitch */
1119: #define PICA 0x000
1120: #define ELITE 0x400
1121: #define FINE 0x800
1122:
1123: /* PrintQuality */
1124: #define DRAFT 0x000
1125: #define LETTER 0x100
1126:
1127: /* PrintSpacing */
1128: #define SIX_LPI 0x000
1129: #define EIGHT_LPI 0x200
1130:
1131: /* Print Image */
1132: #define IMAGE_POSITIVE 0x00
1133: #define IMAGE_NEGATIVE 0x01
1134:
1135: /* PrintAspect */
1136: #define ASPECT_HORIZ 0x00
1137: #define ASPECT_VERT 0x01
1138:
1139: /* PrintShade */
1140: #define SHADE_BW 0x00
1141: #define SHADE_GREYSCALE 0x01
1142: #define SHADE_COLOR 0x02
1143:
1144: /* PaperSize */
1145: #define US_LETTER 0x00
1146: #define US_LEGAL 0x10
1147: #define N_TRACTOR 0x20
1148: #define W_TRACTOR 0x30
1149: #define CUSTOM 0x40
1150:
1151: /* PrinterType */
1152: #define CUSTOM_NAME 0x00
1153: #define ALPHA_P_101 0x01
1154: #define BROTHER_15XL 0x02
1155: #define CBM_MPS1000 0x03
1156: #define DIAB_630 0x04
1157: #define DIAB_ADV_D25 0x05
1158: #define DIAB_C_150 0x06
1159: #define EPSON 0x07
1160: #define EPSON_JX_80 0x08
1161: #define OKIMATE_20 0x09
1162: #define QUME_LP_20 0x0A
1163: /* new printer entries, 3 October 1985 */
1164: #define HP_LASERJET 0x0B
1165: #define HP_LASERJET_PLUS 0x0C
1166:
1167:
1168:
1169:
1170:
1171: /* =====
*/
1172: /* == Remember == =====

```

```

1173: */
1174: /* -----
1175: /* this structure is used for remembering what memory has
1176: /* been allocated to
1177: /* date by a given routine, so that a premature abort or
1178: /* systematic exit
1179: /* can deallocate memory cleanly, easily, and completely
1180: /*
1181: struct Remember
1182: {
1183: struct Remember *NextRemember;
1184: ULONG RememberSize;
1185: UBYTE *Memory;
1186: };
1187:
1188:
1189: /* -----
1190: /* --- Miscellaneous -----
1191: /* -----
1192: /*
1193: /* = MACROS -----
1194: /*
1195: #define MENUNUM(n) (n & 0x1F)
1196: #define ITEMNUM(n) ((n >> 5) & 0x003F)
1197: #define SUBNUM(n) ((n >> 11) & 0x001F)
1198: #define SHIFTMENU(n) (n & 0x1F)
1199: #define SHIFTPITEM(n) ((n & 0x3F) << 5)
1200: #define SHIFTSUB(n) ((n & 0x1F) << 11)
1201:
1202: /* = MENU STUFF -----
1203: /*
1204: #define NOMENU 0x001F
1205: #define NOITEM 0x003F
1206: #define NOSUB 0x001F
1207: #define MENUNULL 0xFFFF
1208:
1209: /* = -RJ='s peculiarities -----
1210: /*
1211: #define FOREVER for(;;)
1212: #define SIGN(x) (((x) > 0) - ((x) < 0))
1213: #define NOT !
1214:
1215: /* these defines are for the COMMSEQ and CHECKIT menu stuff.
1216: /* If CHECKIT,
1217: /* I'll use a generic Width (for all resolutions) for the
1218: /* CheckMark.
1219: /* If COMMSEQ, likewise I'll use this generic stuff
1220: /*
1221: #define CHECKWIDTH 19
1222: #define COMMWIDTH 27
1223: #define LOWCHECKWIDTH 13
1224:
1225: #define LOWCOMMWIDTH 16
1226:
1227: /* these are the AlertNumber defines. if you are calling
1228: /* DisplayAlert()
1229: /* the AlertNumber you supply must have the ALERT_TYPE bits
1230: /* set to one
1231: /* of these patterns
1232: /*
1233: #define ALERT_TYPE 0x80000000
1234: #define RECOVERY_ALERT 0x00000000 /* the system can recover
1235: /* from this */
1236: #define DEADEND_ALERT 0x80000000 /* no recovery possible,
1237: /* this is it */
1238:
1239: /* When you're defining IntuiText for the Positive and Negative
1240: /* Gadgets
1241: /* created by a call to AutoRequest(), these defines will
1242: /* get you
1243: /* reasonable-looking text. The only field without a define
1244: /* is the IText
1245: /* field; you decide what text goes with the Gadget
1246: /*
1247: #define AUTOFRONTPEN 0
1248: #define AUTOBACKPEN 1
1249: #define AUTODRAWMODE JAM2
1250: #define AUTOLEFTEDGE 6
1251: #define AUTOTOPEDGE 3
1252: #define AUTOITEXTFONT NULL
1253: #define AUTONEXTTEXT NULL
1254:
1255: /* --- RAWMOUSE Codes and Qualifiers (Console OR IDCMP)
1256: /* -----
1257: #define SELECTUP (IECODE_LBUTTON | IECODE_UP_PREFIX)
1258: #define SELECTDOWN (IECODE_LBUTTON)
1259: #define MENUUP (IECODE_RBUTTON | IECODE_UP_PREFIX)
1260: #define MENDOWN (IECODE_RBUTTON)
1261: #define ALTLEFT (IEQUALIFIER_LALT)
1262: #define ALTRIGHT (IEQUALIFIER_RALT)
1263: #define AMIGALEFT (IEQUALIFIER_LCOMMAND)
1264: #define AMIGARIGHT (IEQUALIFIER_RCOMMAND)
1265: #define AMIGAKEYS (AMIGALEFT | AMIGARIGHT)
1266:
1267: #define CURSORUP 0x4C
1268: #define CURSORLEFT 0x4F
1269: #define CURSORRIGHT 0x4E
1270: #define CURSORDOWN 0x4D
1271: #define KEYCODE_Q 0x10
1272: #define KEYCODE_X 0x32
1273: #define KEYCODE_N 0x36
1274: #define KEYCODE_M 0x37
1275:
1276: #endif
1277:
1278:
1279:
1280:
1281:
1282:
1283:
1284:
1285:
1286:
1287:
1288:
1289:
1290:
1291:
1292:
1293:
1294:
1295:
1296:
1297:
1298:
1299:
1300:
1301:
1302:
1303:
1304:
1305:
1306:
1307:
1308:
1309:
1310:
1311:
1312:
1313:
1314:
1315:
1316:
1317:
1318:
1319:
1320:
1321:
1322:
1323:
1324:
1325:
1326:
1327:
1328:
1329:
1330:
1331:
1332:
1333:
1334:
1335:
1336:
1337:
1338:
1339:
1340:
1341:
1342:
1343:
1344:
1345:
1346:
1347:
1348:
1349:
1350:
1351:
1352:
1353:
1354:
1355:
1356:
1357:
1358:
1359:
1360:
1361:
1362:
1363:
1364:
1365:
1366:
1367:
1368:
1369:
1370:
1371:
1372:
1373:
1374:
1375:
1376:
1377:
1378:
1379:
1380:
1381:
1382:
1383:
1384:
1385:
1386:
1387:
1388:
1389:
1390:
1391:
1392:
1393:
1394:
1395:
1396:
1397:
1398:
1399:
1400:
1401:
1402:
1403:
1404:
1405:
1406:
1407:
1408:
1409:
1410:
1411:
1412:
1413:
1414:
1415:
1416:
1417:
1418:
1419:
1420:
1421:
1422:
1423:
1424:
1425:
1426:
1427:
1428:
1429:
1430:
1431:
1432:
1433:
1434:
1435:
1436:
1437:
1438:
1439:
1440:
1441:
1442:
1443:
1444:
1445:
1446:
1447:
1448:
1449:
1450:
1451:
1452:
1453:
1454:
1455:
1456:
1457:
1458:
1459:
1460:
1461:
1462:
1463:
1464:
1465:
1466:
1467:
1468:
1469:
1470:
1471:
1472:
1473:
1474:
1475:
1476:
1477:
1478:
1479:
1480:
1481:
1482:
1483:
1484:
1485:
1486:
1487:
1488:
1489:
1490:
1491:
1492:
1493:
1494:
1495:
1496:
1497:
1498:
1499:
1500:

```



```

1 #ifndef INTUITION_INTUITIONBASE_H
2 #define INTUITION_INTUITIONBASE_H 1
3
4 /** intuitionbase.h ****
5 * Commodore-Amiga, Inc.
6 *
7 * the IntuitionBase structure and supporting structures
8 *
9 * Modification History
10 * date : author : Comments
11 * -----
12 * 3-1-85 --RJ-- created this file!
13 *
14 ****
15
16 #ifndef EXEC_LIBRARIES_H
17 #include "exec/libraries.h"
18 #endif
19
20 #ifndef GRAPHICS_VIEW_H
21 #include "graphics/view.h"
22 #endif
23
24 /*
25 * Be sure to protect yourself against someone modifying these data as
26 * you look at them: This is done by calling:
27 *
28 * lock = LockIBase(0), which returns a ULONG. When done call
29 * UnlockIBase(lock) where lock is what LockIBase() returned.
30 *
31 * NOTE: these library functions are simply stubs now, but should be called
32 * to be compatible with future releases.
33 */
34
35 /* ===== */
36 /* == IntuitionBase == */
37 /* ===== */
38 struct IntuitionBase
39 {
40 struct Library LibNode;
41
42 struct View ViewLord;
43
44 struct Window *ActiveWindow;
45 struct Screen *ActiveScreen;
46
47 /* the FirstScreen variable points to the frontmost Screen. Screens are
48 * then maintained in a front to back order using Screen.NextScreen
49 */
50 struct Screen *FirstScreen; /* for linked list of all screens */
51 };
52
53 #endif

```

## Contents

```

lattice/ctype.h.
lattice/dec.h
lattice/dos.h
lattice/error.h
lattice/fcntl.h
lattice/iosl.h
lattice/limits.h
lattice/math.h
lattice/stdio.h

```

```

Listing of ctype.h
1 /**
3 * This header file defines various ASCII character manipulation macros,
4 * as follows:
5 *
6 * isalpha(c) non-zero if c is alpha
7 * isupper(c) non-zero if c is upper case
8 * islower(c) non-zero if c is lower case
9 * isdigit(c) non-zero if c is a digit (0 to 9)
10 * isxdigit(c) non-zero if c is a hexadecimal digit (0 to 9, A to F,
11 * a to f)
12 * isspace(c) non-zero if c is white space
13 * ispunct(c) non-zero if c is punctuation
14 * isalnum(c) non-zero if c is alpha or digit
15 * isprint(c) non-zero if c is printable (including blank)
16 * isgraph(c) non-zero if c is graphic (excluding blank)
17 * iscntrl(c) non-zero if c is control character
18 * isascii(c) non-zero if c is ASCII
19 * iscsym(c) non-zero if valid character for C symbols
20 * iscsymf(c) non-zero if valid first character for C symbols
21 *
22 **/
23
24 #define _U 1 /* upper case flag */
25 #define _L 2 /* lower case flag */
26 #define _N 4 /* number flag */
27 #define _S 8 /* space flag */
28 #define _P 16 /* punctuation flag */
29 #define _C 32 /* control character flag */
30 #define _B 64 /* blank flag */
31 #define _X 128 /* hexadecimal flag */
32
33 extern char _ctype[]; /* character type table */
34
35 #define isalpha(c) (_ctype[(c)+1]&(_U|_L))
36 #define isupper(c) (_ctype[(c)+1]&_U)
37 #define islower(c) (_ctype[(c)+1]&_L)
38 #define isdigit(c) (_ctype[(c)+1]&_N)
39 #define isxdigit(c) (_ctype[(c)+1]&_X)
40 #define isspace(c) (_ctype[(c)+1]&_S)
41 #define ispunct(c) (_ctype[(c)+1]&_P)
42 #define isalnum(c) (_ctype[(c)+1]&(_U|_L|_N))
43 #define isprint(c) (_ctype[(c)+1]&(_P|_U|_L|_N|_B))
44 #define isgraph(c) (_ctype[(c)+1]&(_P|_U|_L|_N))
45 #define iscntrl(c) (_ctype[(c)+1]&_C)
46 #define isascii(c) ((unsigned)(c)<=127)
47 #define iscsym(c) (isalnum(c) || (((c)&127)==0x5f))
48 #define iscsymf(c) (isalpha(c) || (((c)&127)==0x5f))
49
50 #define toupper(c) (islower(c)?(c)-('a'-'A')):(c)
51 #define tolower(c) (isupper(c)?(c)+('a'-'A')):(c)
52 #define toascii(c) ((c)&127)

```

## Listing of dec.h

```

1 /**
3 * This file contains information used by the decimal arithmetic package.
4 *
5 * A floating decimal number is a byte array consisting of a two-byte
6 * header followed by a byte for each two digits. The header has the
7 * following format:
8 *
9 * Byte 0, bit 7: Set if negative number
10 * Byte 0, bits 0 to 6: Number of digit bytes (array length - 2)
11 * Byte 1 Decimal exponent (-128 to +127)
12 *
13 **/
14
15 #define D_DIG 8 /* Maximum number of digit bytes */
16 #define D_MAX (D_DIG+2) /* Maximum number of bytes */
17
18 extern char I0[], I1[], I2[]; /* Integer constants 0, 1, 2 */
19 extern char D5[], D05[], D005[]; /* Decimal constants 0.5, 0.05, 0.005 */
20 extern char PI[], PID2[], PIM2[]; /* Constants PI, PI/2, PI*2 */
21 extern char E[]; /* Constant E (base of natural logs) */
22 extern char M[]; /* Constant log10(E) */
23 extern char DPR[], RPD[]; /* Degrees per radian, radians per degree */
24 extern char SR10[]; /* Square root of 10 */
25 extern char X[], Y[], Z[]; /* Work areas */
26 extern char X1[], Y1[], Z1[]; /* Work areas */
27
28 extern char FDEDIT; /* Set to include leading dollar sign */
29 extern char EDTYPE; /* Set if last cvfd input was exponential */
30 extern char FDDECP; /* decimal point character */
31 extern char EDMONY; /* money symbol */
32
33 extern char *cvfd(), *cvfdx(), *vcfd(), *vcfdi(), *vcfde(), *vcfddc();
34
35

```

Listing of "lattice/dos.h"

```

/**
 * This header file supplies information needed to interface with the
 * particular operating system and C compiler being used.
 */

/**
 * The following definitions specify the particular C compiler being used.
 *
 * LATTICE Lattice C compiler
 * BDS BDS C compiler
 * BTL Bell Labs C compiler or equivalent
 * MANX MANX Aztec C compiler
 */
#define LATTICE 1

/**
 * The following type definitions take care of the particularly nasty
 * machine dependency caused by the unspecified handling of sign extension
 * in the C language. When converting "char" to "int" some compilers
 * will extend the sign, while others will not. Both are correct, and
 * the unsuspecting programmer is the loser. For situations where it
 * matters, the new type "byte" is equivalent to "unsigned char".
 */
#ifdef LATTICE
typedef char byte;
#endif

#ifdef BDS
#define byte char
#endif

#ifdef BTL
typedef unsigned char byte;
#endif

#ifdef MANX
#define byte char
#endif

/**
 *
 * Miscellaneous definitions
 */
#define SECSIZ 128 /* disk sector size */

/**
 *
 * The following structure is a File Control Block. Operating systems
 * with CPM-like characteristics use the FCB to store information about
 * a file while it is open.
 */
struct FCB

```

```

{
 char fcbdrv; /* drive code */
 char fcbnam[8]; /* file name */
 char fcbext[3]; /* file name extension */
 char fcbexn; /* extent number */
 char fcbs1; /* reserved */
 char fcbs2; /* reserved */
 char fcbrc; /* record count */
 char fcbsys[16]; /* reserved */
 char fcbr; /* current record number */
 short fcbrcc; /* random record number */
 char fcbovf; /* random record overflow */
};

#define FCBSIZ sizeof(struct FCB)

/**
 *
 * The following symbols define the sizes of file names and node names.
 */
#define FNSIZE 30 /* maximum file node name size */
#define FMSIZE 30 /* maximum file name size */

/**
 *
 * The following codes are used to open files in various modes.
 */
#ifdef LATTICE
#define OPENR 0x8000 /* open for reading */
#define OPENW 0x8001 /* open for writing */
#define OPENU 0x8002 /* open for read/write */
#define OPENC 0x8001 /* create and open for writing */
#else
#define OPENR 0
#define OPENW 1
#define OPENU 2
#endif

/**
 *
 * The following structure appears at the beginning (low address) of
 * each free memory block.
 */
struct MELT
{
 struct MELT *fwd; /* points to next free block */
#ifdef SPTR
 unsigned size; /* number of MELTs in this block */
#else
 long size; /* number of MELTs in this block */
#endif
};
#define MELTSIZE sizeof(struct MELT)

```

Listing of error.h

```
1 /**
2 *
3 * The file "/include/libraries/dos.h" contains all the error messages.
4 * Do not use this file.
5 *
6 */
7
8 #include "include/libraries/dos.h"
```

Listing of fcntl.h

```
1 /**
2 *
3 * The following symbols are used for the "open" and "creat" functions.
4 *
5 */
6 #define O_RDONLY 0 /* Read-only value (right byte of mode word) */
7 #define O_WRONLY 1 /* Write-only value */
8 #define O_RDWR 2 /* Read-write value */
9
10 #define O_NDELAY 4 /* Non-blocking I/O flag */
11 #define O_APPEND 8 /* Append mode flag */
12 #define O_CREAT 0x0100 /* File creation flag */
13 #define O_TRUNC 0x200 /* File truncation flag */
14 #define O_EXCL 0x400 /* Exclusive access flag */
15
16 #define O_RAW 0x8000 /* Raw I/O flag (Lattice feature) */
17
18 /**
19 *
20 * The following symbols are used for the "fcntl" function.
21 *
22 */
23 #define F_DUPFD 0 /* Duplicate file descriptor */
24 #define F_GETFD 1 /* Get file descriptor flags */
25 #define F_SETFD 2 /* Set file descriptor flags */
26 #define F_GETFL 3 /* Get file flags */
27 #define F_SETFL 4 /* Set file flags */
```

## Listing of ios1.h

```
1 /**
2 *
3 * The following structure is a UNIX file block that retains information about
4 * a file being accessed via the level 1 I/O functions.
5 */
6 struct UFB
7 {
8 char ufbflg; /* flags */
9 char ufbtyp;
10 int ufbfh; /* file handle */
11 };
12 #define NUFBS 20 /* number of UFBs defined */
13
14 /*
15 *
16 * UFB.ufbflg definitions
17 *
18 */
19 #define UFB_OP 0x80 /* file is open */
20 #define UFB_RA 0x40 /* reading is allowed */
21 #define UFB_WA 0x20 /* writing is allowed */
22 #define UFB_NT 0x10 /* access file with no translation */
23 #define UFB_AP 8 /* append mode flag */
24 #define UFB_NC 4 /* no-close flag */
25
26 /*
27 *
28 * UFB.ufbtyp definitions
29 *
30 */
31 #if MSDOS1
32 #define D_DISK 0
33 #define D_CON 1
34 #define D_PRN 2
35 #define D_AUX 3
36 #define D_NULL 4
37 #endif
```

## Listing of limits.h

```
1 #define HUGE_VAL 1.797693E+308
```

## Listing of math.h

```

1 /**
2 *
3 * Redefine secondary simulation function names to become primary names
4 * for systems without a Numeric Data Processor.
5 *
6 */
7 #ifndef NONDP
8 #define _acos acos
9 #define _asin asin
10 #define _atan atan
11 #define _cos cos
12 #define _cosh cosh
13 #define _cot cot
14 #define _exp exp
15 #define _fabs fabs
16 #define _ldexp ldexp
17 #define _log log
18 #define _log10 log10
19 #define _modf modf
20 #define _pow pow
21 #define _pow2 pow2
22 #define _sin sin
23 #define _sinh sinh
24 #define _sqrt sqrt
25 #define _tan tan
26 #define _tanh tanh
27 #endif
28
29 /**
30 *
31 * Structure to hold information about math exceptions
32 *
33 */
34 struct exception
35 {
36 int type; /* error type */
37 char *name; /* math function name */
38 double arg1, arg2; /* function arguments */
39 double retval; /* proposed return value */
40 };
41
42 /**
43 *
44 * Exception type codes, found in exception.type
45 *
46 */
47 #define DOMAIN 1 /* domain error */
48 #define SING 2 /* singularity */
49 #define OVERFLOW 3 /* overflow */
50 #define UNDERFLOW 4 /* underflow */
51 #define TLOSS 5 /* total loss of significance */
52 #define PLOSS 6 /* partial loss of significance */
53
54 /**
55 *
56 * Error codes generated by basic arithmetic operations (+ - * /)
57 *
58 */

```

```

59 #define EPEUND 1 /* underflow */
60 #define EPEOVF 2 /* overflow */
61 #define EPEZDV 3 /* zero divisor */
62 #define EPENAN 4 /* not a number (invalid operation) */
63
64 /**
65 *
66 * Constants
67 *
68 */
69 #define PI 3.14159265358979323846
70 #define PID2 1.57079632679489661923 /* PI divided by 2 */
71 #define PID4 0.78539816339744830962 /* PI divided by 4 */
72 #define I_PI 0.31830988618379067154 /* Inverse of PI */
73 #define I_PID2 0.63661977236758134308 /* Inverse of PID2 */
74
75 #define HUGE 1.797693e308 /* huge value */
76 #define TINY 2.2e-308 /* tiny value */
77 #define LOHUGE 709.778 /* natural log of huge value */
78 #define LOGTINY -708.396 /* natural log of tiny value */
79
80 /**
81 *
82 * External declarations
83 *
84 */
85 extern int _fperr; /* floating point arithmetic error */
86 extern int errno; /* UNIX error code */
87
88 extern char *ecvt();
89 extern short *seed48();
90 extern int atoi(), matherr();
91 extern long atol(), strtol(), lrand48(), nrand48(), mrand48(), jrand48();
92 extern double atof(), exp(), log(), log10(), pow(), sqrt();
93 extern double floor(), ceil(), fmod(), fabs(), frexp(), ldexp(), modf();
94 extern double sinh(), cosh(), tanh(), sin(), cos(), tan(), cot(), asin(), acos();
95 extern double atan(), atan2(), except();
96 extern double drand48(), erand48();

```

## Listing of stdio.h

```

1 /**
2 *
3 * This header file defines the information used by the standard I/O
4 * package.
5 *
6 **/
7 #define _BUFSIZ 512 /* standard buffer size */
8 #define BUFSIZ 512 /* standard buffer size */
9 #define _NFILE 20 /* maximum number of files */
10
11 struct _iobuf
12 {
13 unsigned char *_ptr; /* current buffer pointer */
14 int _rcnt; /* current byte count for reading */
15 int _wcnt; /* current byte count for writing */
16 unsigned char *_base; /* base address of I/O buffer */
17 char _flag; /* control flags */
18 char _file; /* file number */
19 int _size; /* size of buffer */
20 unsigned char _cbuf; /* single char buffer */
21 char _pad; /* (pad to even number of bytes) */
22 };
23
24 extern struct _iobuf _iob[_NFILE];
25
26 #define _IOREAD 1 /* read flag */
27 #define _IOWRT 2 /* write flag */
28 #define _IONBF 4 /* non-buffered flag */
29 #define _IOMYBUF 8 /* private buffer flag */
30 #define _IOEOF 16 /* end-of-file flag */
31 #define _IOERR 32 /* error flag */
32 #define _IOSTRG 64 /* read-write (update) flag */
33 #define _IORW 128
34
35 #ifndef NULL
36 #if SPTR
37 #define NULL 0 /* null pointer value */
38 #else
39 #define NULL 0L
40 #endif
41 #endif
42 #define FILE struct _iobuf /* shorthand */
43 #define EOF (-1) /* end-of-file code */
44
45 #define stdin (&_iob[0]) /* standard input file pointer */
46 #define stdout (&_iob[1]) /* standard output file pointer */
47 #define stderr (&_iob[2]) /* standard error file pointer */
48
49 #define getc(p) (--(p)->_rcnt>=0? *(p)->_ptr++:_filbf(p))
50 #define getchar() getc(stdin)
51 #define putc(c,p) (--(p)->_wcnt>=0? ((int) (*(p)->_ptr++=(c))):_flsbf((c),p))
52 #define putchar(c) putc(c,stdout)
53 #define feof(p) (((p)->_flag&_IOEOF)!=0)
54 #define ferror(p) (((p)->_flag&_IOERR)!=0)
55 #define fileno(p) (p)->_file
56 #define rewind(fp) fseek(fp,0L,0)
57 #define fflush(fp) _flsbf(-1,fp)
58 #define clearerr(fp) clrerr(fp)

```

```

59
60 FILE *fopen();
61 FILE *freopen();
62 long ftell();
63 char *fgets();
64
65 #define abs(x) ((x)<0?-(x):(x))
66 #define max(a,b) ((a)>(b)?(a):(b))
67 #define min(a,b) ((a)<=(b)?(a):(b))
68

```

## Contents

libraries/diskfont.h  
libraries/dos.h  
libraries/dosextens.h  
libraries/intuition.h  
libraries/mathffp.h  
libraries/translator.h

```
1 #ifndef LIBRARIES_DISKFONT_H
2 #define LIBRARIES_DISKFONT_H
3 /*****
4 /* Commodore-Amiga, Inc. */
5 /* diskfont.h */
6 *****/
7
8 *
9 * diskfont library definitions
10 *
11 *****/
12
13 #ifndef EXEC_NODES_H
14 #include "exec/nodes.h"
15 #endif
16 #ifndef EXEC_LISTS_H
17 #include "exec/lists.h"
18 #endif
19 #ifndef GRAPHICS_TEXT_H
20 #include "graphics/text.h"
21 #endif
22
23 #define MAXFONTPATH 256 /* including null terminator */
24
25 struct FontContents {
26 char fc_FileName[MAXFONTPATH];
27 UWORD fc_YSize;
28 UBYTE fc_Style;
29 UBYTE fc_Flags;
30 };
31
32 #define FCH_ID 0x0f00
33
34 struct FontContentsHeader {
35 UWORD fch_FileID; /* FCH_ID */
36 UWORD fch_NumEntries; /* the number of FontContents elements */
37 /* struct FontContents fch_FC[]; */
38 };
39
40 #define DFH_ID 0x0f80
41 #define MAXFONTNAME 32 /* font name including ".font\0" */
42
43 struct DiskFontHeader {
44 /* the following 8 bytes are not actually considered a part of the */
45 /* DiskFontHeader, but immediately precede it. The NextSegment is */
46 /* supplied by the linker/loader, and the ReturnCode is the code */
47 /* at the beginning of the font in case someone runs it... */
48 /* ULONG dfh_NextSegment; /* actually a BPTR */
49 /* ULONG dfh_ReturnCode; /* MOVEQ #0,D0 : RTS */
50 /* here then is the official start of the DiskFontHeader... */
51 struct Node dfh_DF; /* node to link disk fonts */
52 UWORD dfh_FileID; /* DFH_ID */
53 UWORD dfh_Revision; /* the font revision */
54 LONG dfh_Segment; /* the segment address when loaded */
55 char dfh_Name[MAXFONTNAME]; /* the font name (null terminated) */
56 struct TextFont dfh_TF; /* loaded TextFont structure */
57 };
58
59
```



```

60 #define AFB_MEMORY 0
61 #define AFF_MEMORY 1
62 #define AFB_DISK 1
63 #define AFF_DISK 2
64
65 struct AvailFonts {
66 UWORD af_Type; /* MEMORY or DISK */
67 struct TextAttr af_Attr; /* text attributes for font */
68 };
69
70 struct AvailFontsHeader {
71 UWORD afh_NumEntries; /* number of AvailFonts elements */
72 /* struct AvailFonts afh_AF[]; */
73 };
74
75 #endif

```

```

1: #ifndef LIBRARIES_DOS_H
2: #define LIBRARIES_DOS_H
3: /*****
4: /* Commodore-Amiga, Inc.
5: /* dos.h
6: /* Standard C header for AmigaDOS on the MC68000
7: /*****
8:
9: #ifndef EXEC_TYPES_H
10: #include "exec/types.h"
11: #endif
12:
13: #define DOSNAME "dos.library"
14:
15: /* Predefined Amiga DOS global constants */
16:
17: /* Mode parameter to Open() */
18: #define MODE_OLDFILE 1005 /* Open existing file
19: read/write
20: * positioned at beginning
21: of file. */
22: #define MODE_NEWFILE 1006 /* Open freshly created
23: file (delete
24: * old file) read/write
25: */
26:
27: /* Relative position to Seek() */
28: #define OFFSET_BEGINNING -1 /* relative to Beginning
29: Of File */
30: #define OFFSET_CURRENT 0 /* relative to Current
31: file position */
32: #define OFFSET_END 1 /* relative to End Of
33: File */
34:
35: #define OFFSET_BEGINNING OFFSET_BEGINNING /* ancient
36: compatibility */
37:
38: #define BITSPERBYTE 8
39: #define BYTESPERLONG 4
40: #define BITSPERLONG 32
41: #define MAXINT 0x7FFFFFFF
42: #define MININT 0x80000000
43:
44: /* Passed as type to Lock() */
45: #define SHARED_LOCK -2 /* File is readable
46: by others */
47: #define ACCESS_READ -2 /* Synonym */
48: #define EXCLUSIVE_LOCK -1 /* No other access allowed
49: */
50: #define ACCESS_WRITE -1 /* Synonym */
51:
52: struct DateStamp {
53: LONG ds_Days; /* Number of days since Jan.
54: 1, 1978 */
55: LONG ds_Minute; /* Number of minutes past
56: midnight */

```

```

45: LONG ds_Tick; /* Number of ticks past minute
 */
46: }; /* DateStamp */
47: #define TICKS_PER_SECOND 50 /* Number of ticks in
 one second */
48:
49: /* Returned by Examine() and ExInfo(), must be on a 4 byte
 boundary */
50: struct FileInfoBlock {
51: LONG fib_DiskKey;
52: LONG fib_DirEntryType; /* Type of Directory. If <
 0, then a plain file.
53: * If > 0 a directory */
54: char fib_FileName[108]; /* Null terminated. Max 30
 chars used for now */
55: LONG fib_Protection; /* bit mask of protection,
 rwx are 3-0. */
56: LONG fib_EntryType;
57: LONG fib_Size; /* Number of bytes in file
 */
58: LONG fib_NumBlocks; /* Number of blocks in file
 */
59: struct DateStamp fib_Date; /* Date file last changed */
60: char fib_Comment[116]; /* Null terminated.
61: * Comment associated with
 file */
62: }; /* FileInfoBlock */
63:
64: /* FIB stands for FileInfoBlock */
65: /* FIBB are bit definitions, FIBF are field definitions
 */
66: #define FIBB_READ 3
67: #define FIBB_WRITE 2
68: #define FIBB_EXECUTE 1
69: #define FIBB_DELETE 0
70: #define FIBF_READ (1<<FIBB_READ)
71: #define FIBF_WRITE (1<<FIBB_WRITE)
72: #define FIBF_EXECUTE (1<<FIBB_EXECUTE)
73: #define FIBF_DELETE (1<<FIBB_DELETE)
74:
75:
76: /* All BCPL data must be long word aligned. BCPL pointers
 are the long word
77: * address (i.e byte address divided by 4 (>>2)) */
78: typedef long BPTR; /* Long word pointer
 */
79: typedef long BSTR; /* Long word pointer
 to BCPL string */
80: #define BADDR(bptr) (bptr << 2) /* Convert BPTR to typical
 C pointer */
81: /* BCPL strings have a length in the first byte and then
 the characters.
82: * For example: s[0]=3 s[1]=S s[2]=Y s[3]=S
 */
83:
84: /* returned by Info(), must be on a 4 byte boundary */
85: struct InfoData {
86: LONG id_NumSoftErrors; /* number of soft errors on
 disk */
87: LONG id_UnitNumber; /* Which unit disk is (was)
 mounted on */
88: LONG id_DiskState; /* See defines below */
89: LONG id_NumBlocks; /* Number of blocks on disk
 */
90: LONG id_NumBlocksUsed; /* Number of block in use
 */
91: LONG id_BytesPerBlock;
92: LONG id_DiskType; /* Disk Type code */
93: BPTR id_VolumeNode; /* BCPL pointer to volume
 node */
94: LONG id_InUse; /* Flag, zero if not in use
 */
95: }; /* InfoData */
96:
97: /* ID stands for InfoData */
98: /* Disk states */
99: #define ID_WRITE_PROTECTED 80 /* Disk is write protected
 */
100: #define ID_VALIDATING 81 /* Disk is currently being
 validated */
101: #define ID_INVALIDATED 82 /* Disk is consistent and
 writeable */
102:
103: /* Disk types */
104: #define ID_NO_DISK_PRESENT (-1)
105: #define ID_UNREADABLE_DISK (('B'<<24) | ('A'<<16) | ('D'<<8))
106: #define ID_DOS_DISK (('D'<<24) | ('O'<<16) | ('S'<<8))
107: #define ID_NOT_REALLY_DOS (('N'<<24) | ('D'<<16) | ('O'<<8)
 | ('S'))
108: #define ID_KICKSTART_DISK (('K'<<24) | ('I'<<16) | ('C'<<8)
 | ('K'))
109:
110: /* Errors from IoErr(), etc. */
111: #define ERROR_NO_FREE_STORE 103
112: #define ERROR_NO_DEFAULT_DIR 201
113: #define ERROR_OBJECT_IN_USE 202
114: #define ERROR_OBJECT_EXISTS 203
115: #define ERROR_DIR_NOT_FOUND 204
116: #define ERROR_OBJECT_NOT_FOUND 205
117: #define ERROR_BAD_STREAM_NAME 206
118: #define ERROR_OBJECT_TOO_LARGE 207
119: #define ERROR_ACTION_NOT_KNOWN 209
120: #define ERROR_INVALID_COMPONENT_NAME 210
121: #define ERROR_INVALID_LOCK 211
122: #define ERROR_OBJECT_WRONG_TYPE 212
123: #define ERROR_DISK_NOT_VALIDATED 213
124: #define ERROR_DISK_WRITE_PROTECTED 214
125: #define ERROR_RENAME_ACROSS_DEVICES 215
126: #define ERROR_DIRECTORY_NOT_EMPTY 216
127: #define ERROR_TOO_MANY_LEVELS 217
128: #define ERROR_DEVICE_NOT_MOUNTED 218
129: #define ERROR_SEEK_ERROR 219
130: #define ERROR_COMMENT_TOO_BIG 220
131: #define ERROR_DISK_FULL 221
132: #define ERROR_DELETE_PROTECTED 222
133: #define ERROR_WRITE_PROTECTED 223
134: #define ERROR_READ_PROTECTED 224
135: #define ERROR_NOT_A_DOS_DISK 225
136: #define ERROR_NO_DISK 226
137: #define ERROR_NO_MORE_ENTRIES 232

```

```

138:
139: /* These are the return codes used by convention by AmigaDOS
 commands */
140: /* See FAILAT and IF for relvance to EXECUTE files
 */
141: #define RETURN_OK 0 /* No problems,
 success */
142: #define RETURN_WARN 5 /* A warning
 only */
143: #define RETURN_ERROR 10 /* Something
 wrong */
144: #define RETURN_FAIL 20 /* Complete
 or severe failure*/
145:
146: /* Bit numbers that signal you that a user has issued a
 break */
147: #define SIGBREAKB_CTRL_C 12
148: #define SIGBREAKB_CTRL_D 13
149: #define SIGBREAKB_CTRL_E 14
150: #define SIGBREAKB_CTRL_F 15
151:
152: /* Bit fields that signal you that a user has issued a break
 */
153: /* for example: if (SetSignal(0,0) & BREAK_CTRL_CF) cleanup_and_exit();
 */
154: #define SIGBREAKF_CTRL_C (1<<SIGBREAKB_CTRL_C)
155: #define SIGBREAKF_CTRL_D (1<<SIGBREAKB_CTRL_D)
156: #define SIGBREAKF_CTRL_E (1<<SIGBREAKB_CTRL_E)
157: #define SIGBREAKF_CTRL_F (1<<SIGBREAKB_CTRL_F)
158:
159: #endif LIBRARIES_DOS_H
160:

```

```

1
2 #ifndef LIBRARIES_DOSEXTENS_H
3 #define LIBRARIES_DOSEXTENS_H 1
4 /******
5 /* Commodore-Amiga, Inc.
6 /* dosextens.h
7 /******
8 /* DOS structures not needed for the casual DOS user */
9
10 #ifndef EXEC_TYPES_H
11 #include "exec/types.h"
12 #endif
13 #ifndef EXEC_TASKS_H
14 #include "exec/tasks.h"
15 #endif
16 #ifndef EXEC_PORTS_H
17 #include "exec/ports.h"
18 #endif
19 #ifndef EXEC_LIBRARIES_H
20 #include "exec/libraries.h"
21 #endif
22
23 #ifndef LIBRARIES_DOS_H
24 #include "libraries/dos.h"
25 #endif
26
27 /* All DOS processes have this structure */
28 /* Create and Device Proc returns pointer to the MsgPort in this structure */
29 /* dev_proc = (struct Process *) (DeviceProc(..) - sizeof(struct Task)); */
30
31 struct Process {
32 struct Task pr_Task;
33 struct MsgPort pr_MsgPort; /* This is BPTR address from DOS functions */
34 WORD pr_Pad; /* Remaining variables on 4 byte boundaries */
35 BPTR pr_SegList; /* Array of seg lists used by this process */
36 LONG pr_StackSize; /* Size of process stack in bytes */
37 APTR pr_GlobVec; /* Global vector for this process (BCPL) */
38 LONG pr_TaskNum; /* CLI task number of zero if not a CLI */
39 BPTR pr_StackBase; /* Ptr to high memory end of process stack */
40 LONG pr_Result2; /* Value of secondary result from last call */
41 BPTR pr_CurrentDir; /* Lock associated with current directory */
42 BPTR pr_CIS; /* Current CLI Input Stream */
43 BPTR pr_COS; /* Current CLI Output Stream */
44 APTR pr_ConsoleTask; /* Console handler process for the
45 * current window*/
46 APTR pr_FileSystemTask; /* File handler process for current drive */
47 BPTR pr_CLI; /* pointer to ConsoleLineInterpreter */
48 APTR pr_ReturnAddr; /* pointer to previous stack frame */
49 APTR pr_PktWait; /* Function to be called when awaiting msg */
50 APTR pr_WindowPtr; /* Window for error printing */
51 }; /* Process */
52
53 /* The long word address (BPTR) of this structure is returned by
54 * Open() and other routines that return a file. You need only worry
55 * about this struct to do async io's via PutMsg() instead of
56 * standard file system calls */
57
58 struct FileHandle {
59 struct Message *fh_Link; /* EXEC message */

```

```

60 struct MsgPort *fh_Port; /* Reply port for the packet */
61 struct MsgPort *fh_Type; /* Port to do PutMsg() to
62 * Address is negative if a plain file */
63 LONG fh_Buf;
64 LONG fh_Pos;
65 LONG fh_End;
66 LONG fh_Funcs;
67 #define fh_Funcl fh_Funcs
68 LONG fh_Func2;
69 LONG fh_Func3;
70 LONG fh_Args;
71 #define fh_Arg1 fh_Args
72 LONG fh_Arg2;
73 }; /* FileHandle */
74
75 /* This is the extension to EXEC Messages used by DOS */
76
77 struct DosPacket {
78 struct Message *dp_Link; /* EXEC message */
79 struct MsgPort *dp_Port; /* Reply port for the packet */
80 /* Must be filled in each send. */
81 LONG dp_Type; /* See ACTION... below and
82 * 'R' means Read, 'W' means Write to the
83 * file system */
84 LONG dp_Res1; /* For file system calls this is the result
85 * that would have been returned by the
86 * function, e.g. Write ('W') returns actual
87 * length written */
88 LONG dp_Res2; /* For file system calls this is what would
89 * have been returned by IoErr() */
90 /* Device packets common equivalents */
91 #define dp_Action dp_Type
92 #define dp_Status dp_Res1
93 #define dp_Status2 dp_Res2
94 #define dp_BufAddr dp_Arg1
95 LONG dp_Arg1;
96 LONG dp_Arg2;
97 LONG dp_Arg3;
98 LONG dp_Arg4;
99 LONG dp_Arg5;
100 LONG dp_Arg6;
101 LONG dp_Arg7;
102 }; /* DosPacket */
103
104 /* A Packet does not require the Message to be before it in memory, but
105 * for convenience it is useful to associate the two.
106 * Also see the function init_std_pkt for initializing this structure */
107
108 struct StandardPacket {
109 struct Message sp_Msg;
110 struct DosPacket sp_Pkt;
111 }; /* StandardPacket */
112
113 /* Packet types */
114 #define ACTION_NIL 0
115 #define ACTION_GET_BLOCK 2
116 #define ACTION_SET_MAP 4
117 #define ACTION_DIE 5
118 #define ACTION_EVENT 6
119 #define ACTION_CURRENT_VOLUME 7
120 #define ACTION_LOCATE_OBJECT 8
121 #define ACTION_RENAME_DISK 9
122 #define ACTION_WRITE 'W'
123 #define ACTION_READ 'R'
124 #define ACTION_FREE_LOCK 15
125 #define ACTION_DELETE_OBJECT 16
126 #define ACTION_RENAME_OBJECT 17
127
128 #define ACTION_COPY_DIR 19
129 #define ACTION_WAIT_CHAR 20
130 #define ACTION_SET_PROTECT 21
131 #define ACTION_CREATE_DIR 22
132 #define ACTION_EXAMINE_OBJECT 23
133 #define ACTION_EXAMINE_NEXT 24
134 #define ACTION_DISK_INFO 25
135 #define ACTION_INFO 26
136
137 #define ACTION_SET_COMMENT 28
138 #define ACTION_PARENT 29
139 #define ACTION_TIMER 30
140 #define ACTION_INHIBIT 31
141 #define ACTION_DISK_TYPE 32
142 #define ACTION_DISK_CHANGE 33
143 /* Action Set List */
144 /* DOS library node structure.
145 * This is the data at positive offsets from the library node.
146 * Negative offsets from the node is the jump table to DOS functions
147 * node = (struct DosLibrary *) OpenLibrary("dos.library" ..) */
148
149 struct DosLibrary {
150 struct Library dl_lib;
151 APTR dl_Root; /* Pointer to RootNode, described below */
152 APTR dl_GV; /* Pointer to BCPL global vector */
153 LONG dl_A2; /* Private register dump of DOS */
154 LONG dl_A5;
155 LONG dl_A6;
156 }; /* DosLibrary */
157
158 /*
159
160 struct RootNode {
161 BPTR rn_TaskArray; /* [0] is max number of CLI's
162 * [1] is APTR to process id of CLI 1
163 * [n] is APTR to process id of CLI n */
164 BPTR rn_ConsoleSegment; /* SegList for the CLI */
165 struct DateStamp rn_Time; /* Current time */
166 LONG rn_RestartSeg; /* SegList for the disk validator process */
167 BPTR rn_Info; /* Pointer of the Info structure */
168 }; /* RootNode */
169
170 struct DosInfo {
171 BPTR di_McName; /* Network name of this machine; currently 0 */
172 BPTR di_DevInfo; /* Device List */
173 BPTR di_Devices; /* Currently zero */
174 BPTR di_Handlers; /* Currently zero */
175 APTR di_NetHand; /* Network handler processid; currently zero */
176 }; /* DosInfo */
177
178 /* DOS Processes started from the CLI via RUN or NEWCLI have this additional
179 * set to data associated with them */

```

```

180
181 struct CommandLineInterface {
182 LONG cli_Result2; /* Value of IoErr from last command */
183 BSTR cli_SetName; /* Name of current directory */
184 BPTR cli_CommandDir; /* Lock associated with command directory */
185 LONG cli_ReturnCode; /* Return code from last command */
186 BSTR cli_CommandName; /* Name of current command */
187 LONG cli_FailLevel; /* Fail level (set by FAILAT) */
188 BSTR cli_Prompt; /* Current prompt (set by PROMPT) */
189 BPTR cli_StandardInput; /* Default (terminal) CLI input */
190 BPTR cli_CurrentInput; /* Current CLI input */
191 BSTR cli_CommandFile; /* Name of EXECUTE command file */
192 LONG cli_Interactive; /* Boolean; True if prompts required */
193 LONG cli_Background; /* Boolean; True if CLI created by RUN */
194 BPTR cli_CurrentOutput; /* Current CLI output */
195 LONG cli_DefaultStack; /* Stack size to be obtained in long words */
196 BPTR cli_StandardOutput; /* Default (terminal) CLI output */
197 BPTR cli_Module; /* SegList of currently loaded command */
198 }; /* CommandLineInterface */
199
200 /*
201 * this structure needs some work. It should really be a union, because
202 * it can take on different valued depending on whether it is a device,
203 * an assigned directory, or a volume.
204 * For now, it reflects a volume.
205 */
206 struct DeviceList {
207 BPTR dl_Next; /* bptr to next device list */
208 LONG dl_Type; /* see DLT below */
209 struct MsgPort * dl_Task; /* ptr to handler task */
210 BPTR dl_Lock; /* not for volumes */
211 struct DateStamp dl_VolumeDate; /* creation date */
212 BPTR dl_LockList; /* outstanding locks */
213 LONG dl_DiskType; /* 'DOS', etc */
214 LONG dl_unused;
215 BSTR * dl_Name; /* bptr to bcpl name */
216 };
217
218 /* definitions for dl_Type */
219 #define DLT_DEVICE 0
220 #define DLT_DIRECTORY 1
221 #define DLT_VOLUME 2
222
223
224 /* a lock structure, as returned by Lock() or DupLock() */
225 struct FileLock {
226 BPTR fl_Link; /* bcpl pointer to next lock */
227 LONG fl_Key; /* disk block number */
228 LONG fl_Access; /* exclusive or shared */
229 struct MsgPort * fl_Task; /* handler task's port */
230 BPTR fl_Volume; /* bptr to a DeviceList */
231 };
232
233 #endif LIBRARIES_DOEXTENS_H

```

```

1 #ifndef LIBRARIES_MATHFFP_H
2 #define LIBRARIES_MATHFFP_H
3 /******
4 /* Commodore-Amiga, Inc.
5 /* mathffp.h
6 /******
7
8 /*
9 * general floating point declarations
10 */
11 #define PI ((FLOAT) 3.1415192653857)
12 #define TWO_PI ((FLOAT) 2) * PI
13 #define PI2 (PI / ((FLOAT) 2))
14 #define PI4 (PI / ((FLOAT) 4))
15 #define E ((FLOAT) 2.7182818284590453)
16 #define LOG10 ((FLOAT) 2.3025850929940456)
17
18 #define FPTEN ((FLOAT) 10.0)
19 #define FPONE ((FLOAT) 1.0)
20 #define FPHALF ((FLOAT) 0.5)
21 #define FPZERO ((FLOAT) 0.0)
22
23 #define trunc(x) ((int) (x))
24 #define round(x) ((int) ((x) + 0.5))
25 #define itof(i) ((FLOAT) (i))
26
27 int SPFix(); /* Basic math functions */
28 FLOAT SPFlt();
29 int SPCmp();
30 int SPTst();
31 FLOAT SPAbs(), abs();
32 FLOAT SPNeg();
33 FLOAT SPAdd();
34 FLOAT SPSub();
35 FLOAT SPMul();
36 FLOAT SPDiv();
37
38 FLOAT SPAsin(), SPACos(), SPAtan(); /* Transcendental math functions */
39 FLOAT SPSin(), SPCos(), SPTan(), SPSincos();
40 FLOAT SPSinh(), SPCosh(), SPTanh();
41 FLOAT SPExp(), SPLog(), SPLog10(), SPPow();
42 FLOAT SPSqrt(), SPFieee();
43
44 FLOAT afp(), dbf(); /* Math conversion functions */
45
46 #endif !LIBRARIES_MATHFFP_H

```

```
1 #ifndef LIBRARIES_TRANSLATOR_H
2 #define LIBRARIES_TRANSLATOR_H
3 /*****
4 /* Commodore-Amiga, Inc.
5 /* translator.h
6 /*****
7
8
9 /* Translator error return codes */
10
11 #define TR_NotUsed -1 /* This is an oft used system rc */
12 #define TR_NoMem -2 /* Can't allocate memory */
13 #define TR_MakeBad -4 /* Error in MakeLibrary call */
14
15 #endif LIBRARIES_TRANSLATOR_H
```

Contents

resources/cia.h  
resources/disk.h  
resources/misc.h  
resources/potgo.h

```

1: /*****
2: /* Commodore-Amiga, Inc.
3: /* */
4: /* cia.h
5: /* */
6: /*****
7: #define CIAA_NAME "ciaa.resource"
8: #define CIAB_NAME "ciab.resource"
9:

```

```

1
2 #ifndef RESOURCES_DISK_H
3 #define RESOURCES_DISK_H
4 /*****
5 /* Commodore-Amiga, Inc.
6 /* disk.h
7 /*****
8
9 /*****
10 *
11 * external declarations for disk resources
12 *
13 * SOURCE CONTROL
14 * -----
15 * $Header: disk.h,v 27.2 85/07/12 23:12:44 neil Exp $
16 *
17 * $Locker: $
18 *
19 *****/
20
21 #ifndef EXEC_TYPES_H
22 #include "exec/types.h"
23 #endif !EXEC_TYPES_H
24
25 #ifndef EXEC_LISTS_H
26 #include "exec/lists.h"
27 #endif !EXEC_LISTS_H
28
29 #ifndef EXEC_PORTS_H
30 #include "exec/ports.h"
31 #endif !EXEC_PORTS_H
32
33 #ifndef EXEC_INTERRUPTS_H
34 #include "exec/interrupts.h"
35 #endif !EXEC_INTERRUPTS_H
36
37 #ifndef EXEC_LIBRARIES_H
38 #include "exec/libraries.h"
39 #endif !EXEC_LIBRARIES_H
40
41
42 /*****
43 *
44 * Resource structures
45 *
46 *****/
47
48
49 struct DiscResourceUnit {
50 struct Message dru_Message;
51 struct Interrupt dru_DiscBlock;
52 struct Interrupt dru_DiscSync;
53 struct Interrupt dru_Index;
54 };
55
56 struct DiscResource {
57 struct Library dr_Library;
58 struct DiscResourceUnit *dr_Current;
59 UBYTE dr_Flags;

```

```

60 UBYTE dr_pad;
61 struct Library *dr_SysLib;
62 struct Library *dr_CiaResource;
63 ULONG dr_UnitID[4];
64 struct List dr_Waiting;
65 struct Interrupt dr_DiscBlock;
66 struct Interrupt dr_DiscSync;
67 struct Interrupt dr_Index;
68 };
69
70 /* dr_Flags entries */
71 #define DRB_ALLOC0 0 /* unit zero is allocated */
72 #define DRB_ALLOC1 1 /* unit one is allocated */
73 #define DRB_ALLOC2 2 /* unit two is allocated */
74 #define DRB_ALLOC3 3 /* unit three is allocated */
75 #define DRB_ACTIVE 7 /* is the disk currently busy? */
76
77 #define DRF_ALLOC0 (1<<0) /* unit zero is allocated */
78 #define DRF_ALLOC1 (1<<1) /* unit one is allocated */
79 #define DRF_ALLOC2 (1<<2) /* unit two is allocated */
80 #define DRF_ALLOC3 (1<<3) /* unit three is allocated */
81 #define DRF_ACTIVE (1<<7) /* is the disk currently busy? */
82
83
84
85 /*****
86 *
87 * Hardware Magic
88 *
89 *****/
90
91
92 #define DSKDMAOFF 0x4000 /* idle command for dsklen register */
93
94
95 /*****
96 *
97 * Resource specific commands
98 *
99 *****/
100
101 /*
102 * DISKNAME is a generic macro to get the name of the resource.
103 * This way if the name is ever changed you will pick up the
104 * change automatically.
105 */
106
107 #define DISKNAME "disk.resource"
108
109
110 #define DR_ALLOCUNIT (LIB_BASE - 0*LIB_VECTSIZE)
111 #define DR_FREEUNIT (LIB_BASE - 1*LIB_VECTSIZE)
112 #define DR_GETUNIT (LIB_BASE - 2*LIB_VECTSIZE)
113 #define DR_GIVEUNIT (LIB_BASE - 3*LIB_VECTSIZE)
114 #define DR_GETUNITID (LIB_BASE - 4*LIB_VECTSIZE)
115
116
117 #define DR_LASTCOMM (DR_GIVEUNIT)
118
119 /*****

```

```

120 *
121 * drive types
122 *
123 *****/
124
125 #define DRT_AMIGA (0x00000000)
126 #define DRT_37422D2S (0x55555555)
127 #define DRT_EMPTY (0xFFFFFFFF)
128
129 #endif RESOURCES_DISK_H

```



```

1 #ifndef RESOURCES_MISC_I
2 #define RESOURCES_MISC_I
3 /*****
4 /* Commodore-Amiga, Inc. */
5 /* misc.h */
6 *****/
7
8 /*****
9 *
10 * external declarations for misc system resources
11 *
12 * SOURCE CONTROL
13 * -----
14 * $Header: misc.h,v 27.3 85/07/12 16:28:29 neil Exp $
15 *
16 * $Locker: $
17 *
18 *****/
19
20 #ifndef EXEC_TYPES_H
21 #include "exec/types.h"
22 #endif !EXEC_TYPES_H
23
24 #ifndef EXEC_LIBRARIES_H
25 #include "exec/libraries.h"
26 #endif !EXEC_LIBRARIES_H
27
28
29 /*****
30 *
31 * Resource structures
32 *
33 *****/
34
35 #define MR_SERIALPORT 0
36 #define MR_SERIALBITS 1
37 #define MR_PARALLELPORT 2
38 #define MR_PARALLELBITS 3
39
40 #define NUMMRTYPES 4
41
42 struct MiscResource {
43 struct Library mr_Library;
44 ULONG mr_AllocArray[NUMMRTYPES];
45 };
46
47 #define MR_ALLOCMISCRESOURCE (LIB_BASE)
48 #define MR_FREEMISCRESOURCE (LIB_BASE + LIB_VECSIZE)
49
50
51 #define MISCNAME "misc.resource"
52
53 #endif !RESOURCES_MISC_H

```

```

1 #ifndef RESOURCES_POTGO_H
2 #define RESOURCES_POTGO_H
3 /*****
4 /* Commodore-Amiga, Inc. */
5 /* potgo.h */
6 *****/
7 #define POTGONAME "potgo.resource"
8 #endif

```

Contents

workbench/icon.h  
workbench/startup.h  
workbench/workbench.h

D-177

```
1
2 #ifndef LIBRARIES_ICON_H
3 #define LIBRARIES_ICON_H
4
5 /*****
6 /* Commodore-Amiga, Inc. */
7 /* icon.h */
8 /*****
9
10 /*****
11 *
12 * icon.h -- external declarations for workbench support library
13 *
14 * SOURCE CONTROL
15 * -----
16 * $Header: icon.h,v 31.1 85/08/31 09:10:56 neil Exp $
17 *
18 * $Locker: $
19 *
20 *****/
21
22
23 /*****
24 *
25 * library structures
26 *
27 *****/
28
29
30 #define ICONNAME "icon.library"
31
32 /*****
33 *
34 * function types
35 *
36 *****/
37
38 struct WObject *GetWObject(), *AllocWObject();
39 LONG PutWObject(), PutIcon(), GetIcon(), MatchToolValue();
40 VOID FreeFreeList(), FreeWObject(), AddFreeList();
41 char *ToolTypeArray();
42
43
44 #endif !LIBRARIES_ICON_H
```

```

1 /*****
2 /* Commodore-Amiga, Inc. */
3 /* startup.h */
4 /*****
5
6 /* NOTE: This file is NOT used to generate lib/Astartup.obj or */
7 /* lib/Lstartup.obj. */
8
9 #ifndef EXEC_TYPES_H
10 #include "exec/types.h"
11 #endif !EXEC_TYPES_H
12
13 #ifndef EXEC_PORTS_H
14 #include "exec/ports.h"
15 #endif !EXEC_PORTS_H
16
17 #ifndef LIBRARIES_DOS_H
18 #include "libraries/dos.h"
19 #endif !LIBRARIES_DOS_H
20
21 struct WBStartup {
22 struct Message sm_Message; /* a standard message structure */
23 struct MsgPort * sm_Process; /* the process descriptor for you */
24 BPTR sm_Segment; /* a descriptor for your code */
25 LONG sm_NumArgs; /* the number of elements in ArgList */
26 char * sm_ToolWindow; /* description of window */
27 struct WBArg * sm_ArgList; /* the arguments themselves */
28 };
29
30 struct WBArg {
31 BPTR wa_Lock; /* a lock descriptor */
32 BYTE * wa_Name; /* a string relative to that lock */
33 };
34

```

```

1 /*****
2 *
3 * workbench.h
4 *
5 * Commodore-Amiga, Inc.
6 *
7 * $Header: workbench.h,v 31.4 85/10/27 13:50:28 neil Exp $
8 *
9 * $Locker: $
10 *
11 *
12 *****/
13
14 #ifndef EXEC_TYPES_H
15 #include "exec/types.h"
16 #endif !EXEC_TYPES_H
17
18 #ifndef EXEC_NODES_H
19 #include "exec/nodes.h"
20 #endif !EXEC_NODES_H
21
22 #ifndef EXEC_LISTS_H
23 #include "exec/lists.h"
24 #endif !EXEC_LISTS_H
25
26 #ifndef EXEC_TASKS_H
27 #include "exec/tasks.h"
28 #endif !EXEC_TASKS_H
29
30 #ifndef INTUITION_INTUITION_H
31 #include "intuition/intuition.h"
32 #endif !INTUITION_INTUITION_H
33
34 #define WBDISK 1
35 #define WBDRAWER 2
36 #define WBTOOL 3
37 #define WBPROJECT 4
38 #define WBGARBAGE 5
39 #define WBDEVICE 6
40 #define WBKICK 7
41
42 struct DrawerData {
43 struct NewWindow dd_NewWindow; /* args to open window */
44 LONG dd_CurrentX; /* current x coordinate of origin */
45 LONG dd_CurrentY; /* current y coordinate of origin */
46 LONG dd_MinX; /* smallest x coordinate in window */
47 LONG dd_MinY; /* smallest y coordinate in window */
48 LONG dd_MaxX; /* largest x coordinate in window */
49 LONG dd_MaxY; /* largest y coordinate in window */
50 struct Gadget dd_HorizScroll;
51 struct Gadget dd_VertScroll;
52 struct Gadget dd_UpMove;
53 struct Gadget dd_DownMove;
54 struct Gadget dd_LeftMove;
55 struct Gadget dd_RightMove;
56 struct Image dd_HorizImage;
57 struct Image dd_VertImage;
58 struct PropInfo dd_HorizProp;
59 struct PropInfo dd_VertProp;

```

```

60 struct Window * dd_DrawerWin; /* pointer to drawers window */
61 struct WObject * dd_Object; /* back pointer to drawer object */
62 struct List dd_Children; /* where our children hang out */
63 LONG dd_Lock;
64 };
65
66 /* the amount of DrawerData actually written to disk */
67 #define DRAWERDATAFILESIZE (sizeof(struct NewWindow) + 2*sizeof(LONG))
68
69
70 struct DiskObject {
71 UWORD do_Magic; /* a magic number at the start of the file*/
72 UWORD do_Version; /* a version number, so we can change it*/
73 struct Gadget do_Gadget; /* a copy of in core gadget */
74 UBYTE do_Type;
75 char * do_DefaultTool;
76 char ** do_ToolTypes;
77 LONG do_CurrentX;
78 LONG do_CurrentY;
79 struct DrawerData * do_DrawerData;
80 char * do_ToolWindow; /* only applies to tools */
81 LONG do_StackSize; /* only applies to tools */
82
83 };
84
85 #define WB_DISKMAGIC 0xe310 /* a magic number, not easily impersonated */
86 #define WB_DISKVERSION 1 /* our current version number */
87
88 struct FreeList {
89 WORD fl_NumFree;
90 struct List fl_MemList;
91 };
92
93 struct WObject {
94 struct Node wo_MasterNode; /* all objects are on this list */
95 struct Node wo_Siblings; /* list of drawer members */
96 struct Node wo_SelectNode; /* list of all selected objects */
97 struct Node wo_UtilityNode; /* function specific linkages */
98 struct WObject * wo_Parent;
99
100 /* object flags */
101 #ifdef SMARTCOMPILER
102 UBYTE wo_IconDisp:l; /* icon is currently in a window */
103 UBYTE wo_DrawerOpen:l; /* we're a drawer, and it is open */
104 UBYTE wo_Selected:l; /* our icon is selected */
105 UBYTE wo_Background:l; /* set if icon is in background */
106 #else
107 /* lattice is not full system V compatible (yet)... */
108 UBYTE wo_Flags;
109 #endif
110
111 UBYTE wo_Type; /* what flavor object is this? */
112 USHORT wo_UseCount; /* number of references to this
113 object */
114 char * wo_Name; /* this object's textual name */
115 SHORT wo_NameXOffset; /* where to put the name */
116 SHORT wo_NameYOffset;
117
118 char * wo_DefaultTool;
119 struct DrawerData * wo_DrawerData; /* if this is a drawer or disk */
120
121 struct Window * wo_IconWin; /* each object's icon lives here */
122 LONG wo_CurrentX; /* virtual X in drawer */
123 LONG wo_CurrentY; /* virtual Y in drawer */
124 char ** wo_ToolTypes; /* the types for this tool */
125 struct Gadget wo_Gadget; /* NOT a pointer, but an instance
126 of a gadget structure */
127 struct FreeList wo_FreeList; /* this objects free list */
128 char * wo_ToolWindow; /* character string for tool's
129 window */
130 LONG wo_StackSize; /* how much stack to give to this */
131 LONG wo_Lock; /* if this tool is in the backdrop */
132 };
133
134 #define TMalloc(size, type) ((type)MALLOC(size))
135 #define ObjAlloc(obj, size, type) ((type)OALLOC(obj, size))
136 #define STREQ(a, b) (!strcmp(a, b))
137
138 /* each message that comes into the WorkBenchPort must have a type field
139 * in the preceding short. These are the defines for this type
140 */
141
142 #define MTYPE_PSTD 1 /* a "standard Potion" message */
143 #define MTYPE_TOOLEXIT 2 /* exit message from our tools */
144 #define MTYPE_DISKCHANGE 3 /* dos telling us of a disk change */
145 #define MTYPE_TIMER 4 /* we got a timer tick */
146 #define MTYPE_CLOSEDOWN 5 /* <unimplemented> */
147 #define MTYPE_IOPROC 6 /* <unimplemented> */
148
149 /* we use the gadget id field to encode some special information */
150 #define GID_WBOBJECT 0 /* a normal workbench object */
151 #define GID_HORIZONTAL 1 /* the horizontal scroll gadget for a drawer */
152 #define GID_VERTICAL 2 /* the vertical scroll gadget for a drawer */
153 #define GID_LEFTSCROLL 3 /* move one window left */
154 #define GID_RIGHTSCROLL 4 /* move one window right */
155 #define GID_UPSCROLL 5 /* move one window up */
156 #define GID_DOWNSCROLL 6 /* move one window down */
157 #define GID_NAME 7 /* the name field for an object */
158
159
160 /* workbench does different complement modes for its gadgets.
161 * It supports separate images, complement mode, and backfill mode.
162 * The first two are identical to intuitions GADGIMAGE and GADGHCOMP.
163 * backfill is similar to GADGHCOMP, but the region outside of the
164 * image (which normally would be color three when complemented)
165 * is flood-filled to color zero.
166 */
167 #define GADGBACKFILL 0x0001
168
169 /* if an icon does not really live anywhere, set its current position
170 * to here
171 */
172 #define NO_ICON_POSITION (0x80000000)

```



# Appendix E

## Printer Device Source Code

This appendix contains the printer-dependent source code for the following printers:

hpplus - Hewlett Packard LaserJet Plus

okimate20 - Okidata

epson - Epson X-80 series

diablo\_c - Diablo C-150

In addition, this appendix includes the following:

- o *macros.i*, which is required in order to assemble any of the “.asm” files
- o *prtbase.h*, which contains printer data structure definitions
- o a document called *Amiga Printer Support Information*, which contains additional information about supported printers and supported features, standard cables for printers, and standard switch settings for printers.

The files in this appendix are intended to aid developers in creating their own custom printer drivers that can be added to the DEVS: directory on an AmigaDOS disk. The documentation that explains the contents of these files is in the “Printer Device” chapter of this manual.

The sequence of linking the various files together is critical. Here is a sample command to ALINK that specifies the files in the correct sequence. Note that the drive specifiers given in this sample link command simply reflect the disks on which the various files were placed and do not necessarily reflect your development environment.

```
ALINK DF1:lib/Astartup.obj+DF0:printertag.obj+DF0:init.obj+
DF0:data.o+DF0:dospecial.o+DF0:render.o+DF0:wait.obj
library DF1:lib/amiga.lib+DF1:lib/lc.lib TO
DF0:printer.ld
```

```

SECTION printer
*----- Included Files -----
INCLUDE "exec/types.i"
INCLUDE "exec/ports.i"
INCLUDE "exec/devices.i"
INCLUDE "exec/io.i"

INCLUDE "devices/timer.i"

*(INCLUDE "macros.i")

* printer device macro definitions

*----- external definition macros -----
XREF_EXE MACRO
XREF _LVO\1
 ENDM

XREF_DOS MACRO
XREF _LVO\1
 ENDM

XREF_GFX MACRO
XREF _LVO\1
 ENDM

XREF_ITU MACRO
XREF _LVO\1
 ENDM

*----- library dispatch macros -----
CALLEXE MACRO
CALLLIB _LVO\1
 ENDM

LINKEXE MACRO
LINKLIB _LVO\1,_SysBase
 ENDM

LINKDOS MACRO
LINKLIB _LVO\1,_DOSBase
 ENDM

LINKGFX MACRO
LINKLIB _LVO\1,_GfxBase
 ENDM

LINKITU MACRO
LINKLIB _LVO\1,_IntuitionBase
 ENDM

INCLUDE "devices/prtbase.i"

```

```

*----- Imported Functions -----
XREF_EXE Forbid
XREF_EXE Permit
XREF_EXE WaitIO
XREF _SysBase

XREF _PD

*----- Exported Functions -----
XDEF _PWait

*----- printer.device/PWait -----
*
* NAME
* PWait - wait for a time
*
* SYNOPSIS
* PWait(seconds, microseconds);
*
* FUNCTION
* PWait uses the timer device to wait after writes are complete
*
*-----
_PWait:
MOVEM.L A4/A6,-(A7)
MOVE.L _PD,A4
MOVE.L pd_PBothReady(A4),A0
JSR (A0)
TST.L D0
BNE.S error

LEA pd_TIOR(A4),A1
MOVE.W #TR_ADDREQUEST,IO_COMMAND(A1)
MOVE.L 12(A7),IOTV_TIME+TV_SECS(A1)
MOVE.L 16(A7),IOTV_TIME+TV_MICRO(A1)
CLR.B IO_FLAGS(A1)
MOVE.L IO_DEVICE(A1),A6
JSR DEV_BEGINIO(A6)
LINKEXE Forbid
LEA pd_TIOR(A4),A1
LINKEXE WaitIO
LINKEXE Permit
MOVEQ #0,D0
TST.L D0

error:
MOVEM.L (A7)+,A4/A6
RTS
END

```



```

/*****
/* Commodore-Amiga, Inc. */
/* prtbase.h */
/*****
*
* printer device data definition
*
*****/

```

```

#ifdef DEVICES_PRTBASE_H
#define DEVICES_PRTBASE_H

```

```

#ifdef EXEC_NODES_H
#include "exec/nodes.h"
#endif
#ifdef EXEC_LISTS_H
#include "exec/lists.h"
#endif
#ifdef EXEC_PORTS_H
#include "exec/ports.h"
#endif
#ifdef EXEC_LIBRARIES_H
#include "exec/libraries.h"
#endif
#ifdef EXEC_TASKS_H
#include "exec/tasks.h"
#endif

```

```

#ifdef DEVICES_PARALLEL_H
#include "devices/parallel.h"
#endif
#ifdef DEVICES_SERIAL_H
#include "devices/serial.h"
#endif
#ifdef DEVICES_TIMER_H
#include "devices/timer.h"
#endif
#ifdef LIBRARIES_DOSEXTENS_I
#include "libraries/dosexpens.h"
#endif
#ifdef INTUITION_INTUITION_H
#include "intuition/intuition.h"
#endif

```

```

struct DeviceData {
 struct Library dd_Device; /* standard library node */
 APTR dd_Segment; /* A0 when initialized */
 APTR dd_ExecBase; /* A6 for exec */
 APTR dd_CmdVectors; /* command table for device commands */
 APTR dd_CmdBytes; /* bytes describing which command queue */
 UWORD dd_NumCommands; /* the number of commands supported */
};

```

```

#define P_STKSIZE 0x800

```

```

struct PrinterData {
 struct DeviceData pd_Device;
 struct MsgPort pd_Unit; /* the one and only unit */
 BPTR pd_PrinterSegment; /* the printer specific segment */
 UWORD pd_PrinterType; /* the segment printer type */
 struct PrinterSegment *pd_SegmentData; /* the segment data structure */
 UBYTE *pd_PrintBuf; /* the raster print buffer */
 int (*pd_PWrite)(); /* the write function */
 int (*pd_PBothReady)(); /* write function's done */
 union {
 struct IOExtPar pd_p0; /* port I/O request 0 */
 struct IOExtSer pd_s0;
 } pd_ior0;
#define pd_PIOR0 pd_ior0.pd_p0
#define pd_SIOR0 pd_ior0.pd_s0
 union { /* and 1 for double buffering */
 struct IOExtPar pd_p1;
 struct IOExtSer pd_s1;
 } pd_ior1;
#define pd_PIOR1 pd_ior1.pd_p1
#define pd_SIOR1 pd_ior1.pd_s1
 struct timerequest pd_TIOR; /* timer I/O request */
 struct MsgPort pd_IORPort; /* and message reply port */
 struct Task pd_TC; /* write task */
 UBYTE pd_Stk[P_STKSIZE]; /* and stack space */
 UBYTE pd_Flags; /* device flags */
 UBYTE pd_pad;
 struct Preferences pd_Preferences; /* the latest preferences */
 UBYTE pd_PWaitEnabled; /* wait function switch */
};

```

```

#define PPCB_GFX 0
#define PPCF_GFX 0x01
#define PPCB_COLOR 1
#define PPCF_COLOR 0x02

```

```

#define PPC_BWALPHA 0 /* black&white alphanumerics */
#define PPC_BWGFY 1 /* black&white graphics */
#define PPC_COLORGFY 3 /* color graphics */

```

```

#define PCC_BW 1 /* only black&white */
#define PCC_YMC 2 /* only yellow/magenta/cyan */
#define PCC_YMC_BW 3 /* yellow/magenta/cyan or black&white */
#define PCC_YMCB 4 /* yellow/magenta/cyan/black */

```

```

#define PCC_4COLOR 0x4 /* a flag for YMCB and BGRW */
#define PCC_ADDITIVE 0x8 /* not yellow/magenta/cyan/black, */
/* but blue/green/red/white */
#define PCC_WB 0x9 /* only black&white, 0 == BLACK */
#define PCC_BGR 0xa /* blue/green/red */
#define PCC_BGR_WB 0xb /* blue/green/red or black&white */
#define PCC_BGRW 0xc /* blue/green/red/white */

```

```

struct PrinterExtendedData {
 char *ped_PrinterName; /* printer name, null terminated */
 VOID (*ped_Init)(); /* called after LoadSeg */
 VOID (*ped_Expunge)(); /* called before UnLoadSeg */
 VOID (*ped_Open)(); /* called at OpenDevice */
 VOID (*ped_Close)(); /* called at CloseDevice */
 UBYTE ped_PrinterClass; /* printer class */
};

```

```

UBYTE ped_ColorClass; /* color class */
UBYTE ped_MaxColumns; /* number of print columns available */
UBYTE ped_NumCharSets; /* number of character sets */
UWORD ped_NumRows; /* number of raster rows in a raster dump */
ULONG ped_MaxXDots; /* number of dots maximum in a raster dump */
ULONG ped_MaxYDots; /* number of dots maximum in a raster dump */
UWORD ped_XDotsInch; /* horizontal dot density */
UWORD ped_YDotsInch; /* vertical dot density */
char ***ped_Commands; /* printer text command table */
VOID (*ped_DoSpecial)(); /* special command handler */
VOID (*ped_Render)(); /* raster render function */
LONG ped_TimeoutSecs; /* good write timeout */
/* the following only exists if the segment version is 33 or greater */
char **ped_8BitChars; /* conversion strings for the extended font */
};

```

```

struct PrinterSegment {
 ULONG ps_NextSegment; /* (actually a BPTR) */
 ULONG ps_runAlert; /* MOVEQ #0,D0 : RTS */
 UWORD ps_Version; /* segment version */
 UWORD ps_Revision; /* segment revision */
 struct PrinterExtendedData ps_PED; /* printer extended data */
};
#endif

```

E-5

Listing of ddiablo\_c/data.c

```

/* diablo C-150 command table */

/***** printer.device/printers/Diablo_C-150_functions *****/
*
* NAME
* Diablo C-150 functions implemented:
*
* aRIS, aIND, aNEL,
* aSLPP, aLMS, aRMS,
* aHTS, aTBC0, aTBC3, aTBCALL, aTBSALL
*
* special functions implemented:
* aRIN, aSLRM, aSFC, aSBC
*
*****/

char *CommandTable[]={
 "\375\033\015P\375", /*reset RIS ESCc */
 "\377", /*initialize*/
 "\012", /* lf IND ESCD */
 "\015\012", /* return,lf NEL ESC */
 "\377", /* reverse lf RI ESCM */

 "\377", /*normal char set SGR 0 ESC[0m */
 "\377", /*italics on SGR 3 ESC[3m */
 "\377", /*italics off SGR 23 ESC[23m */
 "\377", /*underline on SGR 4 ESC[4m */
 "\377", /*underline off SGR 24 ESC[24m */
 "\377", /*boldface on SGR 1 ESC[1m */
 "\377", /*boldface off SGR 22 ESC[22m */
 "\377", /* set foreground color */
 "\377", /* set background color */

 "\377", /*normal space DECSHOP ESC[0w */
 "\377", /*elite on DECSHOP ESC[2w */
 "\377", /*elite off DECSHOP ESC[1w */
 "\377", /* fine on */
 "\377", /* fine off */
 "\377", /*enlarged on GSM (special) */
 "\377", /*enlarged off GSM (special) */

 "\377", /*shadow print on*/
 "\377", /*shadow print off*/
 "\377", /*doublestrike on*/
 "\377", /*doublestrike off*/
 "\377", /* NLQ on*/
 "\377", /* NLQ off*/

 "\377", /*superscript on PLU ESCL */
 "\377", /*superscript off PLD (special) */
 "\377", /*subscript on PLD ESCK */
 "\377", /*subscript off PLU (special) */
 "\377", /* normalize */
 "\377", /* partial line up PLU ESCL */
 "\377", /* partial line down PLD ESCK */

```

```

"\377", /*US char set ESC(B */
"\377", /*French char set ESC(R */
"\377", /*German char set ESC(K */
"\377", /*UK char set ESC(A */
"\377", /*Danish I char set ESC E */
"\377", /*Sweden char set ESC(H */
"\377", /*Italian char set FNT 6 */
"\377", /*Spanish char set FNT 7 */
"\377", /*Japanese char set FNT 8 */
"\377", /*Norweigen char set FNT 9 */
"\377", /*Danish II char set*/

"\377", /*proportional on */
"\377", /*proportional off*/
"\377", /*proportional clear*/
"\377", /*set prop offset TSS */
"\377", /*auto left justify JFY 5 */
"\377", /*auto right justify JFY 7 */
"\377", /*auto full justify JFY 3,6 */
"\377", /*auto justify off JFY 0 */
"\377", /*place holder */
"\377", /*auto center on JFY 2,6 */

"\377", /* 1/8" line space DECVERP ESC[0z */
"\377", /* 1/6" line spacing DECVERP ESC[lz */
"\033\014", /* set form length DECSLPP ESC[Pnt */
"\377", /* perf skip n */
"\377", /* perf skip off */

"\0339", /* Left margin set DECSLRM ESC[Pnl;Pn2s */
"\0330", /* Right margin set */
"\377", /* Top margin set DECSTBM ESC[Pnl;Pn2r */
"\377", /* Bottom marg set */
"\377", /* T&B margin set STBM ESC[Pnl;Pn2r */
"\377", /* L&R margin set SLRM ESC[Pnl;Pn2s */
"\377", /* Clear margins */
"\03315\015\033r90\015",

"\0331", /* Set horiz tab HTS ESCH */
"\377", /* Set vertical tab VTS ESCJ */
"\0338", /* Clr horiz tab TBC 0 ESC0g */
"\0332", /* Clear all h tabs TBC 3 ESC3g */
"\377", /* Clr vertical tab TBC 1 ESC1g */
"\377", /* Clr all v tabs TBC 4 ESC4g */
"\0332", /* Clr all h & v tabs */
/* set default tabs */
"\033i9,17,25,33,41,49,57,65,73,81,89,97,105,113,121,129",
"\377" /* extended commands */

```

};

## Listing of diablo\_c/dospecial.c

```

/* diablo C-150 special printer functions */

/***** printer.device/printers/Diablo_C-150_special_functions *****/
*
* NAME
* Diablo C-150 special functions implemented:
*
*****/

#include "exec/types.h"
#include "devices/printer.h"
#include "devices/prtbase.h"

extern struct PrinterData *PD;

DoSpecial(command,outputBuffer,vline,currentVMI,crlfFlag,Parms)
char outputBuffer[];
UWORD *command;

BYTE *vline;
UBYTE *currentVMI; /* used for color on this printer */
BYTE *crlfFlag;
UBYTE Parms[];

{
int x=0;
int y=0;
static BYTE ISOcolorTable[10]= {49,51,53,52,55,50,54,48,49,49};
static unsigned char initMarg[]="\033100\015\033r00\015";

if(*command==aRIN) {
currentVMI=0x70; / white background, black text */
outputBuffer[x++]='\015';
outputBuffer[x++]='\012';

Parms[0]=(PD->pd_Preferences.PrintLeftMargin);
Parms[1]=(PD->pd_Preferences.PrintRightMargin);
*command=aSLRM;
}
if(*command==aSLRM) {
Parms[0]=Parms[0]+4;
if(Parms[0]<5)Parms[0]=5;

Parms[1]=Parms[1]+5;
if(Parms[1]>90)Parms[1]=90;

initMarg[2]=(char)((Parms[0]/10)+'0');
initMarg[3]=(char)((Parms[0]-(UBYTE)(Parms[0]/10)*10)+'0');
initMarg[7]=(char)((Parms[1]/10)+'0');
initMarg[8]=(char)((Parms[1]-(UBYTE)(Parms[1]/10)*10)+'0');
while(y<10)outputBuffer[x++]=initMarg[y++];
return(x);
}
if(*command==aSFC)
{
if(Parms[0]==39)Parms[0]=30; /* set defaults */
if(Parms[0]==49)Parms[0]=47;

```

```

if(Parms[0]<40) *currentVMI=((*currentVMI)&240)+(Parms[0]-30);
else *currentVMI=((*currentVMI)&15)+((Parms[0]-40)*16);

outputBuffer[x++]='\033';
outputBuffer[x++]='@';
outputBuffer[x++]=ISOcolorTable[(*currentVMI)&15];
outputBuffer[x++]=ISOcolorTable[(((currentVMI)&240)/16)];
return(x);
}

if(*command==aRIS) PD->pd_PWaitEnabled=253;

return(0);
}

```

Listing of diablo\_c/init.asm

```

TTL '$Header: init.asm,v 1.1 85/10/09 19:27:06 kodiak Exp $'

*
* Copyright 1985, Commodore-Amiga Inc. All rights reserved.
* No part of this program may be reproduced, transmitted,
* transcribed, stored in retrieval system, or translated into
* any language or computer language, in any form or by any
* means, electronic, mechanical, magnetic, optical, chemical,
* manual or otherwise, without the prior written permission of
* Commodore-Amiga Incorporated, 983 University Ave. Building #D,
* Los Gatos, California, 95030
*

*
* printer device functions
*
* Source Control
* -----
* $Header: init.asm,v 1.1 85/10/09 19:27:06 kodiak Exp $
* $Locker: $
* $Log: init.asm,v $
* Revision 1.1 85/10/09 19:27:06 kodiak
* remove _stdout variable
* Revision 1.0 85/10/09 19:23:00 kodiak
* added to rcs for updating in version 1
* Revision 1.0 85/09/25 18:31:27 kodiak
* added to rcs for updating in version 1
* Revision 25.0 85/06/16 01:01:22 kodiak
* added to rcs
*

SECTION printer
*----- Included Files -----
INCLUDE "exec/types.i"
INCLUDE "exec/nodes.i"
INCLUDE "exec/lists.i"
INCLUDE "exec/memory.i"
INCLUDE "exec/ports.i"
INCLUDE "exec/libraries.i"

INCLUDE "macros.i"

*----- Imported Functions -----
XREF_EXE CloseLibrary
XREF_EXE OpenLibrary
XREF _AbsExecBase

XREF _PEDData

*----- Exported Globals -----

```

```

XDEF _Init
XDEF _Expunge
XDEF _Open
XDEF _Close
XDEF _PD
XDEF _PED
XDEF _SysBase
XDEF _DOSBase
XDEF _GfxBase
XDEF _IntuitionBase

```

```

SECTION printer,DATA
_PD DC.L 0
_PED DC.L 0
_SysBase DC.L 0
_DOSBase DC.L 0
_GfxBase DC.L 0
_IntuitionBase DC.L 0

```

```

SECTION printer,CODE
_Init:

```

```

MOVE.L 4(A7),_PD
LEA _PEDData(PC),A0
MOVE.L A0,_PED
MOVE.L A6,-(A7)
MOVE.L _AbsExecBase,A6
MOVE.L A6,_SysBase

```

```

* ;----- open the dos library
LEA DLName(PC),A1
MOVEQ #0,D0
CALLEXE OpenLibrary
MOVE.L D0,_DOSBase
BEQ initDLerr

```

```

* ;----- open the graphics library
LEA GLName(PC),A1
MOVEQ #0,D0
CALLEXE OpenLibrary
MOVE.L D0,_GfxBase
BEQ initGLErr

```

```

* ;----- open the intuition library
LEA ILName(PC),A1
MOVEQ #0,D0
CALLEXE OpenLibrary
MOVE.L D0,_IntuitionBase
BEQ initILErr

```

```
MOVEQ #0,D0
```

```
pdIRts:
MOVE.L (A7)+,A6
RTS

```

```
initPAErr:
```

```
MOVE.L _IntuitionBase,A1
LINKEXE CloseLibrary

```

```
initILErr:
```

```
MOVE.L _GfxBase,A1
LINKEXE CloseLibrary

```

```
initGLErr:
```

```
MOVE.L _DOSBase,A1
LINKEXE CloseLibrary

```

```
initDLerr:
```

```
MOVEQ #-1,D0
BRA.S pdIRts

```

```
ILName:
```

```
DC.B 'intuition.library'
DC.B 0

```

```
DLName:
```

```
DC.B 'dos.library'
DC.B 0

```

```
GLName:
```

```
DC.B 'graphics.library'
DC.B 0
DS.W 0

```

```
-----*
```

```
_Expunge:
```

```
MOVE.L _IntuitionBase,A1
LINKEXE CloseLibrary

```

```
MOVE.L _GfxBase,A1
LINKEXE CloseLibrary

```

```
MOVE.L _DOSBase,A1
LINKEXE CloseLibrary

```

```
-----*
```

```
_Open:
```

```
MOVEQ #0,D0
RTS

```

```
-----*
```

```
_Close:
```

```
MOVEQ #0,D0
RTS

```

```
END
```

Listing of diablo\_c/printertag.asm

INCLUDE "exec/strings.i"

INCLUDE "devices/prtbase.i"

TTL '\$Header: printertag.asm,v 32.1 86/02/10 14:32:33 kodiak Exp \$'

\*\*\*\*\*
\*
\* Copyright 1985, Commodore-Amiga Inc. All rights reserved.
\* No part of this program may be reproduced, transmitted,
\* transcribed, stored in retrieval system, or translated into
\* any language or computer language, in any form or by any
\* means, electronic, mechanical, magnetic, optical, chemical,
\* manual or otherwise, without the prior written permission of
\* Commodore-Amiga Incorporated, 983 University Ave. Building #D,
\* Los Gatos, California, 95030
\*
\*\*\*\*\*

\*----- Imported Names -----
XREF \_Init
XREF \_Expunge
XREF \_Open
XREF \_Close
XREF \_CommandTable
XREF \_PrinterSegmentData
XREF \_DoSpecial
XREF \_Render

printer device dependent code tag

\*----- Exported Names -----
XDEF \_PEDData

Source Control

\$Header: printertag.asm,v 32.1 86/02/10 14:32:33 kodiak Exp \$

\*\*\*\*\*

\$Locker: \$

MOVEQ #0,D0 ; show error for OpenLibrary()
RTS
DC.W VERSION
DC.W REVISION

\$Log: printertag.asm,v \$

Revision 32.1 86/02/10 14:32:33 kodiak
add null 8BitChars field

\_PEDData:
DC.L printerName
DC.L \_Init
DC.L \_Expunge
DC.L \_Open
DC.L \_Close
DC.B PPC\_COLORGFX ; PrinterClass
DC.B PCC\_YMCB ; ColorClass
DC.B 80 ; MaxColumns
DC.B 1 ; NumCharSets
DC.W 4 ; NumRows
DC.L 1024 ; MaxXDots
DC.L 0 ; MaxYDots
DC.W 120 ; XDotsInch
DC.W 120 ; YDotsInch
DC.L \_CommandTable ; Commands
DC.L \_DoSpecial
DC.L \_Render
DC.L 60 ; twice normal: slow alpha
DC.L 0 ; 8BitChars

Revision 32.0 86/02/10 14:22:17 kodiak
added to rcs for updating

Revision 1.2 85/10/09 23:57:10 kodiak
replace reference to pdata w/ prtbase

Revision 1.1 85/09/25 18:45:12 kodiak
double timeout: alpha is too slow to print 400 chars in 30 sec.

Revision 1.0 85/09/25 18:32:57 kodiak
added to rcs for updating in version 1

Revision 25.1 85/06/16 01:02:15 kodiak
\*\*\* empty log message \*\*\*

Revision 25.0 85/06/15 06:40:00 kodiak
added to rcs

Revision 25.0 85/06/13 18:53:36 kodiak
added to rcs

printerName:
STRING <'Diablo C-150'>

END

SECTION printer

\*----- Included Files -----

INCLUDE "exec/types.i"
INCLUDE "exec/nodes.i"

Listing of diablo\_c/render.c

```

/*****
#include <exec/types.h>
#include <exec/nodes.h>
#include <exec/lists.h>
#include <exec/memory.h>
#include "devices/printer.h"
#include "devices/prtbase.h"

extern struct PrinterData *PD;
extern struct PrinterExtendedData *PED;

/* for the DIABLO C-150 */
int Render(ct, x, y, status) /* passed a color type */
 UBYTE ct; /* the color type to use (0, 1, 2 or 3) */
 UWORD x, y; /* the x & y co-ordinates */
 UBYTE status; /* print status (0-init, 1-enter pixel, 2-dump) */
{
 static UWORD ROWSIZE, ROWSIZES[4];
 static UWORD COLORSIZE;
 static UWORD BUFSIZE;
 static UWORD colors[4]; /* color ptrs */
 static BYTE huns,tens,ones, center; /* used to program buffer size */
 static UWORD bufptr; /* for double buffering; points to buffer 1 or 2 */
 static UBYTE *ptr, bit_table[] = {128, 64, 32, 16, 8, 4, 2, 1};
 UWORD i; /* mics. var */
 BYTE err; /* the error # */

 switch(status)
 {
 case 0 : /* alloc memory for printer buffer (uses double buffering) */
 i = (center) ? ((PED->ped_MaxXDots - x) / 2) : 0;
 /* get # of centering pixels */
 ROWSIZE=(x+i+7)/8; /* pc/8 pixels per row on the DIABLO C-150 */
 huns=ROWSIZE/100;
 tens=(ROWSIZE-huns*100)/10;
 ones=(ROWSIZE-huns*100-tens*10);
 ROWSIZE += 7; /* plus 7 cmd bytes */
 COLORSIZE=(ROWSIZE*4); /*the size of each color buffer */
 BUFSIZE=(COLORSIZE*4+3);
 /* buffer size required for DIABLO C-150 */
 i = (i+7) / 8; /* convert to byte offset */
 colors[0] = 7 + i; /* black */
 colors[1] = COLORSIZE*2+7 + i; /* yellow */
 colors[2] = COLORSIZE+7 + i; /* magenta */
 colors[3] = COLORSIZE*3+7 + i; /* cyan */
 for (i=0; i<4; i++) ROWSIZES[i] = i * ROWSIZE;
 /* compute ROWSIZES */
 PD->pd_PrintBuf = (UBYTE *)
 AllocMem(BUFSIZE*2, MEMF_PUBLIC); /* alloc public mem */
 if (err=(PD->pd_PrintBuf==0)) return(err);
 if (err=(*(PD->pd_PWrite))("\033\rP",3)) return(err);
 /* reset printer to power-up */
 if (err=PWait(1,0)) return(err);
 if (err=(*(PD->pd_PWrite))("\033l5\r",4)) return(err);
 /* set l margin to .5 inch. */
 if (err=(*(PD->pd_PWrite))("\033r90\r",5)) return(err);

```

E-10

```

/* set r margin to 9 inch. */
bufptr=0; /* init to first buffer */
return(0); /* flag all ok */
break;

case 1 : /* put pixel in buffer (called a max of 16384
 * times/print cycle) */
 i = bufptr+x/8+(y&3)*ROWSIZE+colors[ct];
 /* calc which byte to use */
 PD->pd_PrintBuf[i] = PD->pd_PrintBuf[i] | (1 << (7-(x&7)));
 /* fill print buffer */
 PD->pd_PrintBuf[bufptr+(x>>3)+ROWSIZES[y&3]+colors[ct]]
 |= bit_table[x&7]; /* fill print buffer */
 return(0); /* flag all ok */
 break;

case 2 : /* dump buffer to printer */
 if (err=(*(PD->pd_PWrite))(&(PD->pd_PrintBuf[bufptr]),
 BUFSIZE)) return(err);
 bufptr=BUFSIZE-bufptr; /* switch to other buffer */
 return(0); /* flag all ok */
 break;

case 3 : /* clear and init buffer */
 for (i=bufptr; i<BUFSIZE+bufptr; i++)
 PD->pd_PrintBuf[i] = 0; /* clear buffer */
 ptr = &PD->pd_PrintBuf[bufptr];
 i = BUFSIZE;
 while(i-->0) *ptr++ = 0; /* clear buffer */
 for (i=0; i<16; i++) {
 PD->pd_PrintBuf[bufptr+i*ROWSIZE] = 27;
 PD->pd_PrintBuf[bufptr+i*ROWSIZE+1] = 'g';
 PD->pd_PrintBuf[bufptr+i*ROWSIZE+2] = i+'0';
 PD->pd_PrintBuf[bufptr+i*ROWSIZE+3] = huns + '0';
 PD->pd_PrintBuf[bufptr+i*ROWSIZE+4] = tens + '0';
 PD->pd_PrintBuf[bufptr+i*ROWSIZE+5] = ones + '0';
 PD->pd_PrintBuf[bufptr+i*ROWSIZE+6] = ',';
 /* select # of bytes for each line */
 }
 PD->pd_PrintBuf[bufptr+BUFSIZE-3] = 27;
 PD->pd_PrintBuf[bufptr+BUFSIZE-2] = 'k';
 PD->pd_PrintBuf[bufptr+BUFSIZE-1] = 'l';
 return(0); /* flag all ok */
 break;

case 4 : /* free the print buffer memory */
 err=(*(PD->pd_PBothReady));
 /* wait for both buffers to be clear */
 FreeMem(PD->pd_PrintBuf, BUFSIZE*2);
 /* free the print buffers memory */
 return(err); /* return status */
 break;

case 5 : /* io_special flags call */
 center = x & SPECIAL_CENTER; /* set center flag */
 return(0); /* flag all ok */
 break;

default : return(0); /* flag all ok */
}

```

Listing of epson/data.c

```

/* epson X80 series */

/***** printer.device/printers/Epson_functions *****/
*
* Epson X-80 functions implemented:
* aRIS, aIND, aNEL, aSGR0, aSGR3, aSGR23, aSGR4, aSGR24, aSGR1, aSGR22,
* aSHORP0, aSHORP1, aSHORP2, aSHORP3, aSHORP4, aSHORP5, aSHORP6,
* aDEN1, aDEN2, aDEN3, aDEN4,
* aSUS0, aSUS1, aSUS2, aSUS3, aSUS4,
* aFNT0, aFNT1, aFNT2, aFNT3, aFNT4, aFNT5, aFNT6, aFNT7, aFNT8
* aFNT9, aFNT10,
* aPROP1, aPROP2, aJFY5, aJFY7, aJFY6, aJFY0, aJFY3, aJFY2,
* aVERP0, aVERP1, aSLPP, aPERF, aPERF0,
* aTBC3, aTBC4, aTBCALL, aTBSALL
* special functions implemented:
* aRIN, aSUS0, aSUS1, aSUS2, aSUS3, aSUS4,
* aPLU, aPLD, aVERP0, aVERP1, aSLRM, aIND, aCAM
*
*****/

char *CommandTable[] ={
 "\0375\033@\0375", /*reset RIS ESCc */
 "\0377", /*initialize*/
 "\012", /* lf IND ESCD */
 "\015\012", /* return,lf NEL ESCE */
 "\0377", /* reverse lf RI ESCM */

 /*normal char set SGR 0 ESC[0m */
 "\0335\033-\0376\033F",
 "\0334", /*italics on SGR 3 ESC[3m */
 "\0335", /*italics off SGR 23 ESC[23m */
 "\033-\001", /*underline on SGR 4 ESC[4m */
 "\033-\0376", /*underline off SGR 24 ESC[24m */
 "\033E", /*boldface on SGR 1 ESC[1m */
 "\033F", /*boldface off SGR 22 ESC[22m */
 "\0377", /* set foreground color */
 "\0377", /* set background color */

 /* normal char set SHORP ESC[0w */
 "\033P\022\033W\0376",
 "\033M", /*elite on SHORP ESC[2w */
 "\033P", /*elite off SHORP ESC[1w */
 "\017", /*condensed(fine) on SHORP ESC[4w */
 "\022", /*condensed off SHORP ESC[3w */
 "\033W\001", /*enlarged on SHORP ESC[6w */
 "\033W\0376", /*enlarged off SHORP ESC[5w */

 "\0377", /*shadow print on DEN6 ESC[6"z */
 "\0377", /*shadow print off DEN5 ESC[5"z */
 "\033G", /*doublestrike on DEN4 ESC[4"z */
 "\033H", /*doublestrike off DEN3 ESC[3"z */
 "\033x\001", /* NLQ on DEN2 ESC[2"z */
 "\033x\0376", /* NLQ off DEN1 ESC[1"z */

 "\033S\0376", /*superscript on ESC[2u */
 "\033T", /*superscript off ESC[1u */

```



```

"\033S\001", /*subscript on ESC[4u */
"\033T", /*subscript off ESC[3u */
"\033T", /*normalize ESC[0u */
"\377", /* partial line up PLU ESCL */
"\377", /* partial line down PLD ESCK */

"\033R\376", /*US char set FNT0 ESC(B */
"\033R\001", /*French char set FNT1 ESC(R */
"\033R\002", /*German char set FNT2 ESC(K */
"\033R\003", /*UK char set FNT3 ESC(A */
"\033R\004", /*Danish I char set FNT4 ESC E */
"\033R\005", /*Sweden char set FNT5 ESC(H */
"\033R\006", /*Italian char set FNT6 ESC(Y */
"\033R\007", /*Spanish char set FNT7 ESC(Z */
"\033R\010", /*Japanese char set FNT8 ESC(J */
"\033R\011", /*Norwegian char set FNT9 ESC(6 */
"\033R\012", /*Danish II char set FNT10 ESC(C */

"\033pl", /*proportional on PROP ESC[2p */
"\033p0", /*proportional off PROP ESC[lp */
"\377", /*proportional clear PROP ESC[0p */
"\377", /*set prop offset TSS */
"\033x\001\033a\376", /*auto left justify JFY5 ESC[5 F */
"\033x\001\033a\002", /*auto right justify JFY7 ESC[7 F */
"\033x\001\033a\003", /*auto full justify JFY6 ESC[6 F */
"\033a\376", /*auto justify/center off ESC[0 F */
"\377", /*place holder JFY3 ESC[3 F */
"\033x\001\033a\001", /*auto center on JFY2 ESC[2 F */

"\0330", /* 1/8" line space VERP ESC[0z */
"\0332", /* 1/6" line spacing VERP ESC[1z */
"\033C", /* set form length SLPP ESC[Pnt */
"\033N", /* perf skip n ESC[nq */
"\0330", /* perf skip off ESC[0q */

"\377", /* Left margin set ESC[2x */
"\377", /* Right margin set ESC[3x */
"\377", /* top margin set ESC[4x */
"\377", /* Bottom marg set ESC[5x */
"\377", /* T&B margin set STEM ESC[Pnl;Pn2r */
"\377", /* L&R margin set SLRM ESC[Pnl;Pn2s */
"\377", /* Clear margins ESC[0x */

"\377", /* Set horiz tab HTS ESCH */
"\377", /* Set vertical tab VTS ESCJ */
"\377", /* Clr horiz tab TBC 0 ESC[0g */
"\033D\376", /* Clear all h tabs TBC 3 ESC[3g */
"\377", /* Clr vertical tab TBC 1 ESC[1g */
"\377", /* Clr all v tabs TBC 4 ESC[4g */
"\033D\376", /* Clr all h & v tabs ESC#4 */
"\33D\010\020\030\040\050\060\070\100\110\120\130\376",
/*set default tabs */
"\377" /* extended command */

```

};

## Listing of epson/dospecial.c

```

/* epson X80 special commands */

/***** printer.device/printers/Epson_special_functions *****/
*
* NAME
* Epson X80 special functions
*
*****/

#include "exec/types.h"
#include "devices/printer.h"
#include "devices/prtbase.h"

extern struct PrinterData *PD;

DoSpecial(command,outputBuffer,vline,currentVMI,crLfFlag,Parms)
char outputBuffer[];
UWORD *command;
BYTE *vline;
BYTE *currentVMI;
BYTE *crLfFlag;
UBYTE Parms[];

{
 int x=0;
 int y=0;
 static char initMarg[]="\375\033L\033Q\375";
 static char
 initThisPrinter[]="\033x\001\0332\022\0335\033P\033-\376\033F\n\033W";

 if(*command==aRIN) {
 while(x<18) {outputBuffer[x]=initThisPrinter[x];x++;}
 outputBuffer[x++]='\000';
 outputBuffer[12]='\000';

 if((PD->pd_Preferences.PrintQuality)==DRAFT)outputBuffer[2]='\000';

 currentVMI=36; / assume 1/6 line spacing */
 if((PD->pd_Preferences.PrintSpacing)==EIGHT_LPI) { /* wrong again */
 outputBuffer[4]='0';
 *currentVMI=27;
 }
 if ((PD->pd_Preferences.PrintPitch) != PICA)outputBuffer[x++]='\033';
 if((PD->pd_Preferences.PrintPitch)==ELITE)outputBuffer[x++]='M';
 else if((PD->pd_Preferences.PrintPitch)==FINE)
 outputBuffer[x++]='\017';

 Parms[0]=(PD->pd_Preferences.PrintLeftMargin);
 Parms[1]=(PD->pd_Preferences.PrintRightMargin);
 *command=aSLRM;
 }

 if(*command==aSLRM) {
 PD->pd_PWaitEnabled=253; /* wait after this character */
 }
}

```

```

 if(Parms[0]==0)initMarg[3]=0;
 else initMarg[3]=Parms[0]-1;
 initMarg[6]=Parms[1];
 while(y<8)outputBuffer[x++]=initMarg[y++];
 return(x);
}

if(*command==aCAM) {
 PD->pd_PWaitEnabled=253;
 initMarg[3]=0;
 if(PD->pd_Preferences.PrintPitch == FINE)initMarg[6]=96;
 else if(PD->pd_Preferences.PrintPitch == ELITE)initMarg[6]=137;
 else initMarg[6]=80;
 while(y<8)outputBuffer[x++]=initMarg[y++];
 return(x);
}

if(*command==aPLU) {
 if((*vline)==0){(*vline)=1; *command=aSUS2; return(0);}
 if((*vline)<0){(*vline)=0; *command=aSUS3; return(0);}
 return(-1);
}

if(*command==aPLD) {
 if((*vline)==0){(*vline)=-1; *command=aSUS4; return(0);}
 if((*vline)>0){(*vline)=0; *command=aSUS1; return(0);}
 return(-1);
}

if(*command==aSUS0) *vline=0;
if(*command==aSUS1) *vline=0;
if(*command==aSUS2) *vline=1;
if(*command==aSUS3) *vline=0;
if(*command==aSUS4) *vline=-1;

if(*command==aVERP0) *currentVMI=27;

if(*command==aVERP1) *currentVMI=36;

if(*command==aIND) {
 outputBuffer[x++]='\033';
 outputBuffer[x++]='J';
 outputBuffer[x++] = *currentVMI;
 return(x);
}

if(*command==aRIS) PD->pd_PWaitEnabled=253;

return(0);

```

## Listing of epson/init.asm

```

TTL '$Header: init.asm,v 1.1 85/10/09 19:27:14 kodiak Exp $'

*
* Copyright 1985, Commodore-Amiga Inc. All rights reserved.
* No part of this program may be reproduced, transmitted,
* transcribed, stored in retrieval system, or translated into
* any language or computer language, in any form or by any
* means, electronic, mechanical, magnetic, optical, chemical,
* manual or otherwise, without the prior written permission of
* Commodore-Amiga Incorporated, 983 University Ave. Building #D,
* Los Gatos, California, 95030
*

* printer device functions
*
* Source Control
*
* $Header: init.asm,v 1.1 85/10/09 19:27:14 kodiak Exp $
* $Locker: $
* $Log: init.asm,v $
* Revision 1.1 85/10/09 19:27:14 kodiak
* remove stdout variable
* Revision 1.0 85/10/09 19:23:12 kodiak
* added to rcs for updating in version 1
* Revision 25.0 85/06/16 01:01:22 kodiak
* added to rcs
*

SECTION printer
*----- Included Files -----
INCLUDE "exec/types.i"
INCLUDE "exec/nodes.i"
INCLUDE "exec/lists.i"
INCLUDE "exec/memory.i"
INCLUDE "exec/ports.i"
INCLUDE "exec/libraries.i"

INCLUDE "macros.i"

*----- Imported Functions -----
XREF_EXE CloseLibrary
XREF_EXE OpenLibrary
XREF _AbsExecBase

XREF _PEDData

*----- Exported Globals -----
XDEF _Init
XDEF _Expunge

```

```

XDEF _Open
XDEF _Close
XDEF _PD
XDEF _PED
XDEF _SysBase
XDEF _DOSBase
XDEF _GfxBase
XDEF _IntuitionBase

```

\*\*\*\*\*

```

SECTION printer,DATA
_PD DC.L 0
_PED DC.L 0
_SysBase DC.L 0
_DOSBase DC.L 0
_GfxBase DC.L 0
_IntuitionBase DC.L 0

```

\*\*\*\*\*

```

SECTION printer,CODE
_Init:
 MOVE.L 4(A7),_PD
 LEA _PEDData(PC),A0
 MOVE.L A0,_PED
 MOVE.L A6,-(A7)
 MOVE.L _AbsExecBase,A6
 MOVE.L A6,_SysBase

```

E - 14

```

* ;----- open the dos library
 LEA DLName(PC),A1
 MOVEQ #0,D0
 CALLEXE OpenLibrary
 MOVE.L D0,_DOSBase
 BEQ initDLerr

* ;----- open the graphics library
 LEA GLName(PC),A1
 MOVEQ #0,D0
 CALLEXE OpenLibrary
 MOVE.L D0,_GfxBase
 BEQ initGLErr

* ;----- open the intuition library
 LEA ILName(PC),A1
 MOVEQ #0,D0
 CALLEXE OpenLibrary
 MOVE.L D0,_IntuitionBase
 BEQ initILErr

```

```

pdIRts:
 MOVEQ #0,D0
 MOVE.L (A7)+,A6
 RTS

```

```

initPAErr:
 MOVE.L _IntuitionBase,A1
 LINKEXE CloseLibrary

```

```

initILErr:
 MOVE.L _GfxBase,A1
 LINKEXE CloseLibrary

```

```

initGLErr:
 MOVE.L _DOSBase,A1
 LINKEXE CloseLibrary

```

```

initDLerr:
 MOVEQ #-1,D0
 BRA.S pdIRts

```

```

ILName:
 DC.B 'intuition.library'
 DC.B 0

```

```

DLName:
 DC.B 'dos.library'
 DC.B 0

```

```

GLName:
 DC.B 'graphics.library'
 DC.B 0
 DS.W 0

```

```

*-----
_Expunge:
 MOVE.L _IntuitionBase,A1
 LINKEXE CloseLibrary

 MOVE.L _GfxBase,A1
 LINKEXE CloseLibrary

 MOVE.L _DOSBase,A1
 LINKEXE CloseLibrary

```

```

*-----
_Open:
 MOVEQ #0,D0
 RTS

```

```

*-----
_Close:
 MOVEQ #0,D0
 RTS

 END

```

Listing of epson/printertag.asm

INCLUDE "devices/prtbase.i"

TTL '\$Header: printertag.asm,v 32.2 86/02/12 18:15:55 kodiak Exp \$'

\*----- Imported Names -----

\*\*\*\*\*
\*
\* Copyright 1985, Commodore-Amiga Inc. All rights reserved.
\* No part of this program may be reproduced, transmitted,
\* transcribed, stored in retrieval system, or translated into
\* any language or computer language, in any form or by any
\* means, electronic, mechanical, magnetic, optical, chemical,
\* manual or otherwise, without the prior written permission of
\* Commodore-Amiga Incorporated, 983 University Ave. Building #D,
\* Los Gatos, California, 95030
\*
\*\*\*\*\*

XREF \_Init
XREF \_Expunge
XREF \_Open
XREF \_Close
XREF \_CommandTable
XREF \_PrinterSegmentData
XREF \_DoSpecial
XREF \_Render

\*----- Exported Names -----

\*
\* printer device dependent code tag
\*
\* Source Control
\*
\* \$Header: printertag.asm,v 32.2 86/02/12 18:15:55 kodiak Exp \$
\*
\* \$Locker: \$
\* \$Log: printertag.asm,v \$
\* Revision 32.2 86/02/12 18:15:55 kodiak
\* YDotsInch -> 72
\* Revision 32.1 86/02/10 14:32:42 kodiak
\* add null 8BitChars field
\* Revision 32.0 86/02/10 14:22:38 kodiak
\* added to rcs for updating
\* Revision 1.1 85/10/09 23:57:27 kodiak
\* replace reference to pdata w/ prtbase
\* Revision 1.0 85/10/09 23:57:21 kodiak
\* added to rcs for updating in version 1
\* Revision 29.1 85/08/19 08:32:10 kodiak
\* flag a graphics printer, not BWALPHA
\* Revision 29.0 85/08/19 08:31:06 kodiak
\* added to rcs for updating in version 29
\* Revision 25.1 85/06/16 01:02:15 kodiak
\* \*\*\* empty log message \*\*\*
\*
\* Revision 25.0 85/06/15 06:40:00 kodiak
\* added to rcs
\*
\* Revision 25.0 85/06/13 18:53:36 kodiak
\* added to rcs
\*
\*

XDEF \_PEDData

\*\*\*\*\*

MOVEQ #0,D0 ; show error for OpenLibrary()
RTS
DC.W VERSION
DC.W REVISION
\_PEDData:
DC.L printerName
DC.L \_Init
DC.L \_Expunge
DC.L \_Open
DC.L \_Close
DC.B PPC\_BWGFY ; PrinterClass
DC.B PCC\_BW ; ColorClass
DC.B 80 ; MaxColumns
DC.B 10 ; NumCharSets
DC.W 8 ; NumRows
DC.L 960 ; MaxXDots
DC.L 0 ; MaxYDots
DC.W 120 ; XDotsInch
DC.W 72 ; YDotsInch
DC.L \_CommandTable ; Commands
DC.L \_DoSpecial
DC.L \_Render
DC.L 30
DC.L 0 ; 8BitChars

printerName:
STRING <'Epson'>

END

SECTION printer

\*----- Included Files -----

INCLUDE "exec/types.i"
INCLUDE "exec/nodes.i"
INCLUDE "exec/strings.i"

```

epson/render.c

/*****
#include <exec/types.h>
#include <exec/nodes.h>
#include <exec/lists.h>
#include <exec/memory.h>
#include "devices/printer.h"
#include "devices/prtbase.h"

extern struct PrinterData *PD;
extern struct PrinterExtendedData *PED;

/* for the EPSON */
int Render(ct, x, y, status)
UBYTE ct; /* null for b/w printers */
UWORD x, y; /* the x & y co-ordinates */
/* or the pc & pr print values, or special */
UBYTE status; /* print status (0-init, 1-enter pixel, 2-dump) */
{
 static UWORD ROWSIZE;
 static UWORD BUFSIZE;
 static UWORD bufptr, offset;
 static BYTE center, spacing;
 static UBYTE *ptr, bit_table[] = {128, 64, 32, 16, 8, 4, 2, 1};
 UWORD i; /* mics. var */
 BYTE err; /* the error # */

 switch(status)
 {
 case 0 : /* alloc memory for printer buffer */
 i = (center) ? ((PED->ped_MaxXDots - x) / 2) : 0;
 offset = i + 4;
 ROWSIZE=x+i; /* row size required for EPSON */
 BUFSIZE=(6+ROWSIZE); /* buffer size required for EPSON */
 PD->pd_PrintBuf = (UBYTE *)
 AllocMem(BUFSIZE*2, MEMF_PUBLIC); /* alloc public mem */
 if (err=(PD->pd_PrintBuf == 0)) return(err);
 /* reset printer to power-up state */
 if (err=(*(PD->pd_PWrite))("\033@",2)) return(err);
 if (err=PWait(1,0)) return(err);
 /* special Epson spacing code */
 if (spacing==7) {
 if (err=(*(PD->pd_PWrite))("\0331",2)) return(err);
 /* select 7/72 inch spacing */
 }
 else {
 if (err=(*(PD->pd_PWrite))("\0333\030",3)) return(err);
 /* select 24/216 (8/72) inch spacing */
 }
 /* end of special Epson spacing code */
 if (err=(*(PD->pd_PWrite))("\033U1",3)) return(err);
 /* set unidirect mode */
 bufptr=0;
 return(0); /* flag all ok */
 break;

 case 1 : /* put pixel in buffer */

 i = bufptr+x+4; /* calc which byte to use */
 PD->pd_PrintBuf[i] = PD->pd_PrintBuf[i] | (1 << (7-(y&7)));
 /* fill print buffer */
 PD->pd_PrintBuf[bufptr+x+offset] |= bit_table[y&7];
 return(0); /* flag all ok */
 break;

 case 2 : /* dump buffer to printer */
 if (err=(*(PD->pd_PWrite))(&(PD->pd_PrintBuf[bufptr]),
 BUFSIZE)) return(err);
 bufptr=BUFSIZE-bufptr;
 return(0); /* flag all ok */
 break;

 case 3 : /* clear and init buffer */
 for (i=bufptr; i<bufptr+BUFSIZE; i++)
 PD->pd_PrintBuf[i] = 0; /* clear buffer */
 ptr = &PD->pd_PrintBuf[bufptr];
 i = BUFSIZE;
 while (i--) *ptr++ = 0;
 PD->pd_PrintBuf[bufptr] = 27;
 PD->pd_PrintBuf[bufptr+1] = 'L';
 PD->pd_PrintBuf[bufptr+2] = ROWSIZE & 0xff;
 PD->pd_PrintBuf[bufptr+3] = ROWSIZE >> 8;
 PD->pd_PrintBuf[bufptr+BUFSIZE-2] = 10;
 PD->pd_PrintBuf[bufptr+BUFSIZE-1] = 13;
 return(0); /* flag all ok */
 break;

 case 4 : /* free the print buffer memory */
 err=(*(PD->pd_PWrite))("\033@",2);
 /* reset printer to power-up state */
 if (!err) err=(*(PD->pd_PBothReady));
 /* wait for both buffers to empty */
 FreeMem(PD->pd_PrintBuf, BUFSIZE*2); /* free print buffer's memory */
 return(err); /* return status */
 break;

 case 5 : /* io_special flag call */
 center = x & SPECIAL_CENTER; /* set center flag */
 /* special code for Epson spacing */
 if (PD->pd_Preferences.PaperSize==CUSTOM) {
 PED->ped_YDotsInch = (UWORD)82;
 /* for 7/72 spacing */
 /* (72/7*8 gives 82.3)/ there are 82 dpi in y */
 spacing = 7; /* 7/72 inch spacing */
 }
 else { /* else use default of 8/72 spacing */
 PED->ped_YDotsInch = (UWORD)72;
 /* (72/8*8 gives 72); there are 72 dpi in y */
 spacing = 8; /* 8/72 inch spacing */
 }
 return(0); /* flag all ok */
 break;

 default: return(0);
 }
}

```

Listing for hpplus/data.c

```

/* HP command table */

/***** printer.device/HP_LaserJet_Plus_functions *****/
*
* NAME
* HP LaserJet 2686A functions implemented:
*
* aRIS, aIND, aNEL,
* aSGR0, aSGR3, aSGR23, aSGR4, aSGR24, aSGR1, aSGR22,
* aSHORP0, aSHORP1, aSHORP2, aSHORP3, aSHORP4
* aDEN3, aDEN4, aPLU, aPLD,
* aFNT0, aFNT3, aFNT8,
* aPROP0, aPROP1, aPROP2,
* aVERP0, aVERP1, aPERF, aPERF0, aCAM
*
*****/

```

```

char *CommandTable[]={
 "\375\033E\375", /*reset*/
 "\377", /*initialize*/
 "\012", /* lf IND ESCD */
 "\015\012", /* return,lf NEL ESCE */
 "\033&a-lR", /* reverse lf RI ESCM */

 "\033&d\033(sbs", /*normal char set SGR 0 */
 "\033(sls", /*italics on*/
 "\033(ss", /*italics off*/
 "\033&d", /*underline on*/
 "\033&d@", /*underline off */
 "\033(s5B", /*boldface on*/
 "\033(sB", /*boldface off*/
 "\377", /* set foreground color */
 "\377", /* set background color */

 "\033(s10h1T", /* normal pitch */
 "\033(s12h2T", /* elite on*/
 "\033(s10h1T", /* elite off*/
 "\033(s15H", /* condensed on*/
 "\033(s10H", /* condensed off*/
 "\377", /* enlarged on*/
 "\377", /* enlarged off*/

 "\033(s7B", /*shadow print on*/
 "\033(sB", /*shadow print off*/
 "\033(s3B", /*doublestrike on*/
 "\033(sB", /*doublestrike off*/
 "\377", /* NLQ on*/
 "\377", /* NLQ off*/

 "\377", /*superscript on*/
 "\377", /*superscript off*/
 "\377", /*subscript on*/
 "\377", /*subscript off*/
 "\377", /* normalize */

```

```

"\033&a-.5R", /* partial line up PLU ESCL */
"\033=", /* partial line down PLD ESCK */

"\033(U", /*US char set */
"\033(F", /*French char set*/
"\033(G", /*German char set*/
"\033(LE", /*UK char set*/
"\033(D", /*Danish I char set*/
"\033(S", /*Sweden char set*/
"\033(I", /*Italian char set*/
"\033(1S", /*Spanish char set*/
"\033(8K", /*Japanese char set*/
"\033(D", /*Norweigen char set*/
"\033(D", /*Danish II char set*/

"\033(slP", /*proportional on*/
"\033(sP", /*proportional off*/
"\033(sP", /*proportional clear*/
"\377", /*set prop offset*/
"\377", /*auto left justify on*/
"\377", /*auto right justify on*/
"\377", /*auto full justify on*/
"\377", /*auto justify/center off*/
"\377", /*place holder */
"\377", /*auto center on*/

"\033&l8D", /* 1/8" line space*/
"\033&l6D", /* 1/6" line spacing*/
"\377", /* set form length n */
"\033&llL", /* perf skip n */
"\033&lL", /* Perf skip off */

"\377", /* Left margin set */
"\377", /* Right margin set */
"\377", /* Top margin set */
"\377", /* Bottom marg set */
"\377", /* T&B margin set STBM ESC[Pnl;Pn2r */
"\377", /* L&R margin set SLRM ESC[Pnl;Pn2s */
"\0339", /* Clear margins */

"\377", /* Set horiz tab */
"\377", /* Set vertical tab */
"\377", /* Clr horiz tab */
"\377", /* Clear all h tabs */
"\377", /* Clear vertical tab */
"\377", /* Clr all v tabs TBC 4 */
"\377", /* Clr all h & v tabs */
"\377", /* set default tabs */
"\377", /* extended commands */

```

};

E - 17

Listing for hpplus/density.c

```

/* ***** density.c ***** */
#include <exec/types.h>
#include "devices/prtbase.h"
#include "devices/printer.h"

extern struct PrinterExtendedData *PED;
extern char density[];

SetDensity(level)
UWORD level;
{
 switch (level) {
 case SPECIAL_DENSITY1:
 PED->ped_MaxXDots = 600;
 PED->ped_MaxYDots = 795;
 PED->ped_XDotsInch = PED->ped_YDotsInch = 75;
 density[3] = '0';
 density[4] = '7';
 density[5] = '5';
 break;

 case SPECIAL_DENSITY2:
 PED->ped_MaxXDots = 800;
 PED->ped_MaxYDots = 1060;
 PED->ped_XDotsInch = PED->ped_YDotsInch = 100;
 density[3] = '1';
 density[4] = '0';
 density[5] = '0';
 break;

 case SPECIAL_DENSITY3:
 PED->ped_MaxXDots = 1200;
 PED->ped_MaxYDots = 1590;
 PED->ped_XDotsInch = PED->ped_YDotsInch = 150;
 density[3] = '1';
 density[4] = '5';
 density[5] = '0';
 break;

 case SPECIAL_DENSITY4:
 PED->ped_MaxXDots = 2400;
 PED->ped_MaxYDots = 3180;
 PED->ped_XDotsInch = PED->ped_YDotsInch = 300;
 density[3] = '3';
 density[4] = '0';
 density[5] = '0';
 break;

 default: break;
 }
}

```

Listing for hpplus/dospecial.c

```

/* hp special printer functions */

/***** printer.device/printers/HP_LaserJet_Plus_special_functions *****/
*
* NAME
* HP LaserJet 2686A special functions implemented:
*
* aRIN,
* aSUS0, aSUS1, aSUS2, aSUS3, aSUS4
* aPLU, aPLD, aVERP0, aVERP1,
* aSLPP, aSLRM, aSTBM
*
*****/

#include "exec/types.h"
#include "devices/printer.h"
#include "devices/prtbase.h"

extern struct PrinterData *PD;
UWORD textlength,topmargin;

DoSpecial(command,outputBuffer,vline,currentVMI,crlfFlag,Parms)
char outputBuffer[];
UWORD *command;
BYTE *vline;
BYTE *currentVMI;
BYTE *crlfFlag;
UBYTE Parms[];
{
 int x=0;
 int y=0;
 int j=0;
 static char initThisPrinter[]="\033&d@\033&l6D\033(sb10hpsltul2V";
 static char initMarg[]="\033&a0001000M";
 static char initTMarg[]="\033&l000e000F";
 static char initForm[]="\033&l002e000F";

 if(*command==aRIN) {
 while(x<24){outputBuffer[x]=initThisPrinter[x];x++;}
 if((PD->pd_Preferences.PrintSpacing)==EIGHT_LPI) { /* wrong again */
 outputBuffer[7]='8';
 }

 if((PD->pd_Preferences.PrintPitch)==ELITE) {
 outputBuffer[14]='2';
 outputBuffer[18]='2';
 }

 if((PD->pd_Preferences.PrintPitch)==FINE) {
 outputBuffer[14]='5';
 }

 j=x; /* set the fomlength=textlength, top margin of 2 */
 textlength=PD->pd_Preferences.PaperLength;
 topmargin=2;

 while(y<11)outputBuffer[x++]=initForm[y++];
 }
}

```

```

numberString(textlength,j+7,outputBuffer);
y=0;

Parms[0]=(PD->pd_Preferences.PrintLeftMargin);
Parms[1]=(PD->pd_Preferences.PrintRightMargin);
*command=aSLRM;
}

if(*command==aSLRM) {
 j=x;
 while(y<11)outputBuffer[x++]=initMarg[y++];
 numberString(Parms[0]-1,j+3,outputBuffer);
 numberString(Parms[1]-1,j+7,outputBuffer);
 return(x);
}

if((*command==aSUS2)&&(*vline==0)) {*command=aPLU; *vline=1; return(0);}
if((*command==aSUS2)&&(*vline<0)) {*command=aRI; *vline=1; return(0);}
if((*command==aSUS1)&&(*vline>0)) {*command=aPLD; *vline=0; return(0);}

if((*command==aSUS4)&&(*vline==0)) {*command=aPLD; *vline=(-1); return(0);}
if((*command==aSUS4)&&(*vline>0)) {*command=aIND; *vline=(-1); return(0);}
if((*command==aSUS3)&&(*vline<0)) {*command=aPLU; *vline=0; return(0);}

if(*command==aSUS0)
{
 if(*vline>0) *command=aPLD;
 if(*vline<0) *command=aPLU;
 *vline=0;
 return(0);
}

if(*command==aPLU){(*vline)++; return(0);}

if(*command==aPLD){(*vline)--; return(0);}

if(*command==aSTBM) {

if(Parms[0]== 0)Parms[0]=topmargin;
else topmargin = --Parms[0];

if(Parms[1]== 0)Parms[1]=textlength;
else textlength=Parms[1];

 while(x<11){outputBuffer[x]=initTMarg[x]; x++;}
 numberString(Parms[0],3,outputBuffer);
 numberString(Parms[1]-Parms[0],7,outputBuffer);
 return(x);
}

if(*command==aSLPP) {
 while(x<11){outputBuffer[x]=initForm[x]; x++;}
 numberString(topmargin,3,outputBuffer); /*restore textlength,margin*/
 numberString(textlength,7,outputBuffer);
 return(x);
}

if(*command==aRIS) PD->pd_PWaitEnabled=253;

return(0);

```

```

}

VOID
numberString(Param,x,outputBuffer)
 BYTE Param;
 int x;
 char outputBuffer[];
{
if(Param>199){outputBuffer[x++]='2'; Param-=200;}
else if(Param>99){outputBuffer[x++]='1'; Param-=100;}
else outputBuffer[x++]='0'; /* always return 3 digits */

if(Param>9)outputBuffer[x++]=(BYTE)(Param/10)+'0';
else outputBuffer[x++]='0';

outputBuffer[x++]=(Param%10)+'0';
}

Close()
{
/*(*(PD->pd_PWrite))("\033E",2);*/
/*(PD->pd_PWrite)("\014",1);
*(PD->pd_PBothReady)();
return(0);
}

```



Listing for hpplus/init.asm

```

TTL '$Header: init.asm,v 1.1 85/10/09 19:27:38 kodiak Exp $'

*
* Copyright 1985, Commodore-Amiga Inc. All rights reserved.
* No part of this program may be reproduced, transmitted,
* transcribed, stored in retrieval system, or translated into
* any language or computer language, in any form or by any
* means, electronic, mechanical, magnetic, optical, chemical,
* manual or otherwise, without the prior written permission of
* Commodore-Amiga Incorporated, 983 University Ave. Building #D,
* Los Gatos, California, 95030
*

*
* printer device functions
*
* Source Control
* -----
* $Header: init.asm,v 1.1 85/10/09 19:27:38 kodiak Exp $
* $Locker: $
* $Log: init.asm,v $
* Revision 1.1 85/10/09 19:27:38 kodiak
* remove stdout variable
* Revision 1.0 85/10/09 19:23:53 kodiak
* added to rcs for updating in version 1
* Revision 29.1 85/08/02 16:58:43 kodiak
* remove dummy_Close routine -- it's used to finish print of last page.
* Revision 29.0 85/08/02 16:58:17 kodiak
* added to rcs for updating in version 29
* Revision 25.0 85/06/16 01:01:22 kodiak
* added to rcs
*

SECTION printer
*----- Included Files -----
INCLUDE "exec/types.i"
INCLUDE "exec/nodes.i"
INCLUDE "exec/lists.i"
INCLUDE "exec/memory.i"
INCLUDE "exec/ports.i"
INCLUDE "exec/libraries.i"

INCLUDE "macros.i"
*----- Imported Functions -----
XREF_EXE CloseLibrary
XREF_EXE OpenLibrary
XREF _AbsExecBase

XREF _PEDData

```

E - 20

```

*----- Exported Globals -----
XDEF _Init
XDEF _Expunge
XDEF _Open
XDEF _PD
XDEF _PED
XDEF _SysBase
XDEF _DOSBase
XDEF _GfxBase
XDEF _IntuitionBase

SECTION printer,DATA
_PD DC.L 0
_PED DC.L 0
_SysBase DC.L 0
_DOSBase DC.L 0
_GfxBase DC.L 0
_IntuitionBase DC.L 0

SECTION printer,CODE
_Init:
MOVE.L 4(A7),_PD
LEA _PEDData(PC),A0
MOVE.L A0,_PED
MOVE.L A6,-(A7)
MOVE.L _AbsExecBase,A6
MOVE.L A6,_SysBase

*
;----- open the dos library
LEA DLName(PC),A1
MOVEQ #0,D0
CALLEXE OpenLibrary
MOVE.L D0,_DOSBase
BEQ initDLerr

*
;----- open the graphics library
LEA GLName(PC),A1
MOVEQ #0,D0
CALLEXE OpenLibrary
MOVE.L D0,_GfxBase
BEQ initGLErr

*
;----- open the intuition library
LEA ILName(PC),A1
MOVEQ #0,D0
CALLEXE OpenLibrary
MOVE.L D0,_IntuitionBase
BEQ initILerr

MOVEQ #0,D0

pdIRts:
MOVE.L (A7)+,A6

```

RTS

initPAErr:

MOVE.L \_IntuitionBase,Al  
LINKEXE CloseLibrary

initILErr:

MOVE.L \_GfxBase,Al  
LINKEXE CloseLibrary

initGLErr:

MOVE.L \_DOSBase,Al  
LINKEXE CloseLibrary

initDLerr:

MOVEQ #-1,D0  
BRA.S pdiRts

ILName:

DC.B 'intuition.library'  
DC.B 0

DLName:

DC.B 'dos.library'  
DC.B 0

GLName:

DC.B 'graphics.library'  
DC.B 0  
DS.W 0

\*-----  
\_Expunge:

MOVE.L \_IntuitionBase,Al  
LINKEXE CloseLibrary

MOVE.L \_GfxBase,Al  
LINKEXE CloseLibrary

MOVE.L \_DOSBase,Al  
LINKEXE CloseLibrary

\*-----  
\_Open:

MOVEQ #0,D0  
RTS

END

Listing for hpplus/printertag.asm

```

TTL '$Header: printertag.asm,v 32.1 86/02/10 14:33:17 kodiak Exp $'

*
* Copyright 1985, Commodore-Amiga Inc. All rights reserved.
* No part of this program may be reproduced, transmitted,
* transcribed, stored in retrieval system, or translated into
* any language or computer language, in any form or by any
* means, electronic, mechanical, magnetic, optical, chemical,
* manual or otherwise, without the prior written permission of
* Commodore-Amiga Incorporated, 983 University Ave. Building #D,
* Los Gatos, California, 95030
*

*
* printer device dependent code tag
*
* Source Control
*
*-----
* $Header: printertag.asm,v 32.1 86/02/10 14:33:17 kodiak Exp $
*
* $Locker: $
*
* $Log: printertag.asm,v $
* Revision 32.1 86/02/10 14:33:17 kodiak
* add null 8BitChars field
*
* Revision 32.0 86/02/10 14:23:56 kodiak
* added to rcs for updating
*
* Revision 1.2 85/10/09 23:58:23 kodiak
* replace reference to pdata w/ prtbase
*
* Revision 1.1 85/10/09 16:11:31 kodiak
* daveb density changes
*
* Revision 25.1 85/06/16 01:02:15 kodiak
* *** empty log message ***
*
* Revision 25.0 85/06/15 06:40:00 kodiak
* added to rcs
*
* Revision 25.0 85/06/13 18:53:36 kodiak
* added to rcs
*

```

SECTION printer

\*----- Included Files -----

```

INCLUDE "exec/types.i"
INCLUDE "exec/nodes.i"
INCLUDE "exec/strings.i"

```

```
INCLUDE "devices/prtbase.i"
```

```
*----- Imported Names -----
```

```
XREF _Init
XREF _Expunge
XREF _Open
XREF _Close
XREF _CommandTable
XREF _PrinterSegmentData
XREF _DoSpecial
XREF _Render
```

```
*----- Exported Names -----
```

```
XDEF _PEDData
```

```

```

```
MOVEQ #0,D0 ; show error for OpenLibrary()
RTS
DC.W VERSION
DC.W REVISION
```

```
_PEDData:
```

```
DC.L printerName
DC.L _Init
DC.L _Expunge
DC.L _Open
DC.L _Close
DC.B PPC_BWGFY ; PrinterClass
DC.B PCC_BW ; ColorClass
DC.B 0 ; MaxColumns
DC.B 0 ; NumCharSets
DC.W 1 ; NumRows
DC.L 800 ; MaxXDots
DC.L 1020 ; MaxYDots
DC.W 100 ; XDotsInch
DC.W 100 ; YDotsInch
DC.L _CommandTable ; Commands
DC.L _DoSpecial
DC.L _Render
DC.L 30
DC.L 0 ; 8BitChars
```

```
printerName:
```

```
STRING <'HP LaserJet Plus'>
```

```
END
```

```
Listing for hpplus/render.c
```

```
/*-----
```

```
#include <exec/types.h>
#include <exec/nodes.h>
#include <exec/lists.h>
#include <exec/memory.h>
#include "devices/prtbase.h"
#include "devices/printer.h"
```

```
extern struct PrinterData *PD;
extern struct PrinterExtendedData *PED;
extern SetDensity();
char density[8] = "\033*t100R";
```

```
/* for the HP+ 2686A */
```

```
int Render(ct, x, y, status)
 UBYTE ct; /* null for b/w printers */
 UWORD x, y; /* the x & y co-ordinates */
 UBYTE status; /* print status (0-init, 1-enter pixel,
 * 2-dump, 3-close, 4-end) */
```

```
{
 static UWORD ROWSIZE;
 static UWORD BUFSIZE, offset;
 static BYTE center,huns,tens,ones; /* used to program buffer size */
 static UWORD bufptr; /* used for double buffering;
 * points to buffer 1 or 2 */
 static UBYTE *ptr, bit_table[] = {128, 64, 32, 16, 8, 4, 2, 1};
 UWORD i; /* mics. var */
 BYTE err; /* the error # */
```

```
#ifdef DEBUG
kprintf("hp render(%ld, %ld, %ld, %ld, %ld);\n", ct, x, y, status);
#endif
```

```
switch(status)
{
case 0 : /* alloc memory for printer buffer (uses double buffering) */
i = (center) ? ((PED->ped_MaxXDots - x) / 2) : 0;
offset = (i+7)/8 + 7; /* calc centering offset */
ROWSIZE=(x+7+i)/8; /* row size required for HP */
huns=ROWSIZE/100;
tens=(ROWSIZE-huns*100)/10;
ones=(ROWSIZE-huns*100-tens*10);
BUFSIZE=(ROWSIZE+7); /* buffer size required for HP */
PD->pd_PrintBuf = (UBYTE *)
 AllocMem(BUFSIZE*2, MEMF_PUBLIC); /* alloc public mem */
if (err=(PD->pd_PrintBuf == 0)) return(err);
if (err=*(PD->pd_PWrite)("\033E",2)) return(err);
/* reset printer */
if (err=PWait(1,0)) return(err);
if (err=*(PD->pd_PWrite)(density,7)) return(err);
/* set resolution */
if (err=*(PD->pd_PWrite)("\033*r0A",5)) return(err);
/* start raster gfx */
bufptr=0; /* init to first buffer */
return(0); /* flag all ok */
break;
```

```

case 1 : /* put pixel in buffer */
/*
/*
i = bufptr+x/8+7; /* calc which byte to use */
PD->pd_PrintBuf[i] = PD->pd_PrintBuf[i] | (1 << (7-(x&7)));
/* fill print buffer */
PD->pd_PrintBuf[bufptr+(x>>3)+offset] |= bit_table[x&7];
return(0); /* flag all ok */
break;

case 2 : /* dump buffer to printer */
if (err=*(PD->pd_PWrite>(&(PD->pd_PrintBuf[bufptr]),
BUFSIZE)) return(err);
bufptr=BUFSIZE-bufptr; /* switch to other buffer */
return(0); /* flag all ok */
break;

case 3 : /* clear and init buffer */
for (i=bufptr; i<BUFSIZE+bufptr; i++)
PD->pd_PrintBuf[i] = 0; /* clear buffer */
ptr = &PD->pd_PrintBuf[bufptr];
i = BUFSIZE;
while (i-- *ptr++ = 0;
PD->pd_PrintBuf[bufptr] = 27;
PD->pd_PrintBuf[bufptr+1] = '*';
PD->pd_PrintBuf[bufptr+2] = 'b';
PD->pd_PrintBuf[bufptr+3] = huns + '0';
PD->pd_PrintBuf[bufptr+4] = tens + '0';
PD->pd_PrintBuf[bufptr+5] = ones + '0';
PD->pd_PrintBuf[bufptr+6] = 'W';
return(0); /* flag all ok */
break;

case 4 : /* free the print buffer memory */
/* end raster graphics, unload paper, and reset printer */
err=*(PD->pd_PWrite)("033rB014033E",7);
if (!err) err=*(PD->pd_PBothReady)();
/* wait for both buffers to be clear */
FreeMem(PD->pd_PrintBuf,BUFSIZE*2);
/* free the print buffers memory */
return(err); /* return status */
break;

case 5:
center = x & SPECIAL_CENTER; /* set center flag */
if ((x & SPECIAL_DENSITYMASK) == 0) { /* if use prefs */
if (PD->pd_Preferences.PrintQuality == DRAFT)
SetDensity(SPECIAL_DENSITY2); /* 100 dpi */
else SetDensity(SPECIAL_DENSITY3); /* 150 dpi */
}
else SetDensity(x & SPECIAL_DENSITYMASK);
/* else use SPECIAL */
return(0);
break;

default :
return(0);
break;
}
}

```

Listing for okimate20/data.c

/\* okimate 20 commands \*/

```

/***** printer.device/printers/Okimate_20_functions *****/
*
* Okimate 20 functions implemented:
*
* aRIS, aIND, aNEL, aSGR0, aSGR3, aSGR23, aSGR4, aSGR24
* aSHORP0, aSHORP1, aSHORP2, aSHORP3, aSHORP4, aSHORP5, aSHORP6
* aDEN1, aDEN2, aSUS0, aSUS1, aSUS2, aSUS3, aSUS4,
* aVERP0, aVERP1, aSLPP, aPERF, aPERFO
* special functions implemented:
* aRIN, aRI, aSUS0, aSUS1, aSUS2, aSUS3, aSUS4, aPLU, aPLD
*
*****/
char *CommandTable[] = {
"033w022\033A014\0332\033I\001\033%h\033-\0376\033T\033C\0376\011",
"\0377", /*reset aRIS ESCc */
"\012", /*initialize aRIN */
"\015\012", /* lf IND ESCD */
"\0377", /* return,lf NEL ESCE */
"\0377", /* reverse lf RI ESCM */

"\033%h\033-\0376",
"\033%g", /*normal char set SGR 0 ESC[0m */
"\033%h", /*italics on SGR 3 ESC[3m */
"\033%h", /*italics off SGR 23 ESC[23m */
"\033-\001", /*underline on SGR 4 ESC[4m */
"\033-\0376", /*underline off SGR 24 ESC[24m */
"\0377", /*boldface on SGR 1 ESC[1m */
"\0377", /*boldface off SGR 22 ESC[22m */
"\0377", /* set foreground color */
"\0377", /* set background color */

"\022\033w\0376", /*normal spacing DECSHORP ESC[0w */
"\033:", /*elite on DECSHORP ESC[2w */
"\022", /*elite off DECSHORP ESC[1w */
"\017", /* fine on GSM (special) */
"\022", /* fine off GSM (special) */
"\033w\001", /*enlarged on GSM (special) */
"\033w\0376", /*enlarged off GSM (special) */

"\0377", /*shadow print on*/
"\0377", /*shadow print off*/
"\0377", /*doublestrike on*/
"\0377", /*doublestrike off*/
"\033I\002", /* NLQ on*/
"\033I\001", /* NLQ off*/

"\033S\0376", /*superscript on PLU ESCL */
"\033T", /*superscript off PLD (special) */
"\033S\001", /*subscript on PLD ESCK */
"\033T", /*subscript off PLU (special) */
"\033T", /* normalize */

```

```

"\377", /* partial line up PLU ESCL */
"\377", /* partial line down PLD ESCK */

"\377", /*US char set ESC(B */
"\377", /*French char set ESC(R */
"\377", /*German char set ESC(K */
"\377", /*UK char set ESC(A */
"\377", /*Danish I char set ESC E */
"\377", /*Sweden char set ESC(H */
"\377", /*Italian char set FNT 6 */
"\377", /*Spanish char set FNT 7 */
"\377", /*Japanese char set FNT 8 */
"\377", /*Norweigen char set FNT 9 */
"\377", /*Danish II char set*/

"\377", /*proportional on */
"\377", /*proportional off*/
"\377", /*proportional clear*/
"\377", /*set prop offset TSS */
"\377", /*auto left justify JFY 5 */
"\377", /*auto right justify JFY 7 */
"\377", /*auto full justify JFY 3,6 */
"\377", /*auto justify off JFY 0 */
"\377", /*place holder */
"\377", /*auto center on JFY 2,6 */

"\0330", /* 1/8" line space DECVERP ESC[0z */
"\033A\014\0332",
/* 1/6" line spacing DECVERP ESC[lz */
"\033C", /* set form length DECCLPP ESC[Pnt */
"\033N\001", /* perf skip n */
"\0330", /* perf skip off */

"\377", /* Left margin set DECSLRM ESC[Pnl;Pn2s */
"\377", /* Right margin set */
"\377", /* Top margin set DECSTBM ESC[Pnl;Pn2r */
"\377", /* Bottom marg set */
"\377", /* T&B margin set STBM ESC[Pnl;Pn2r */
"\377", /* L&R margin set SLRM ESC[Pnl;Pn2s */
"\377", /* Clear margins */

"\377", /* Set horiz tab HTS ESCH */
"\377", /* Set vertical tab VTS ESCJ */
"\377", /* Clr horiz tab TBC 0 ESC0g */
"\033D\376", /* Clear all h tabs TBC 3 ESC3g */
"\377", /* Clr vertical tab TBC 1 ESC1g */
"\033D\376", /* Clr all v tabs TBC 4 ESC4g */
"\377", /* Clr all h & v tabs */
"\033D\010\020\030\040\050\060\070\100\110\120\376",
/* set default tabs */
"\377" /* extended command */

```

};

Listing for okimate20/dospecial.c

```

/* okimate 20 special commands */

/***** printer.device/printers/Okimate_20_special_functions *****/
*
* NAME
* Okimate 20 special functions
*
*****/
#include "exec/types.h"
#include "devices/printer.h"
#include "devices/prtbase.h"

extern struct PrinterData *PD;
extern struct PrinterExtendedData *PED;

DoSpecial(command,outputBuffer,vline,currentVMI,crlfFlag,Parms)
char outputBuffer[];
UWORD *command;
BYTE *vline;
BYTE *currentVMI;
BYTE *crlfFlag;
UBYTE Parms[];

{
int x=0;
static char initThisPrinter[]="\033I\001\022\0330\033%h\033-\376\r\033W";

if(*command==aRIN)
{
while(x<15){outputBuffer[x]=initThisPrinter[x];x++;}
outputBuffer[11]='\000';
outputBuffer[x++]='\000';

if((PD->pd_Preferences.PrintQuality)==LETTER)outputBuffer[2]='\002';

if((PD->pd_Preferences.PrintPitch)==ELITE) {
outputBuffer[x++]='\033';
outputBuffer[x++]=':';
}
else if((PD->pd_Preferences.PrintPitch)==FINE)outputBuffer[x++]='\017';

currentVMI=27; / assume 1/8 line spacing */
if((PD->pd_Preferences.PrintSpacing)==SIX_LPI) { /* wrong again */
outputBuffer[x++]='\033';
outputBuffer[x++]='A';
outputBuffer[x++]='\014';
outputBuffer[x++]='\033';
outputBuffer[x++]='2';
*currentVMI=36;
}
}
return(x);
}

if(*command==aPLU) {
if((*vline)==0){(*vline)=1; *command=aSUS2; return(0);}
}

```

Listing for okimate20/init.asm

```

 if((*vline)<0){(*vline)=0; *command=aSUS3; return(0);}
 return(-1);
}

if(*command==aPLD) {
 if((*vline)==0){(*vline)=(-1); *command=aSUS4; return(0);}
 if((*vline)>0){(*vline)=0; *command=aSUS1; return(0);}
 return(-1);
}

if(*command==aSUS0) *vline=0;
if(*command==aSUS1) *vline=0;
if(*command==aSUS2) *vline=1;
if(*command==aSUS3) *vline=0;
if(*command==aSUS4) *vline=(-1);

if(*command==aVERP0) *currentVMI=27;

if(*command==aVERP1) *currentVMI=36;

return(0);
}

```

```

TTL '$Header: init.asm,v 1.2 85/10/09 23:58:49 kodiak Exp $'

*
* Copyright 1985, Commodore-Amiga Inc. All rights reserved.
* No part of this program may be reproduced, transmitted,
* transcribed, stored in retrieval system, or translated into
* any language or computer language, in any form or by any
* means, electronic, mechanical, magnetic, optical, chemical,
* manual or otherwise, without the prior written permission of
* Commodore-Amiga Incorporated, 983 University Ave. Building #D,
* Los Gatos, California, 95030
*

*
* printer device functions
*
* Source Control
*
* $Header: init.asm,v 1.2 85/10/09 23:58:49 kodiak Exp $
*
* $Locker: $
*
* $Log: init.asm,v $
* Revision 1.2 85/10/09 23:58:49 kodiak
* replace reference to pdata w/ prtbase
*
* Revision 1.1 85/10/09 19:27:50 kodiak
* remove _stdout variable
*
* Revision 1.0 85/10/09 19:24:13 kodiak
* added to rcs for updating in version 1
*
* Revision 29.0 85/08/07 22:25:32 kodiak
* added to rcs for updating in version 29
*
* Revision 25.0 85/06/16 01:01:22 kodiak
* added to rcs
*

```

```

SECTION printer
----- Included Files -----
INCLUDE "exec/types.i"
INCLUDE "exec/nodes.i"
INCLUDE "exec/lists.i"
INCLUDE "exec/memory.i"
INCLUDE "exec/ports.i"
INCLUDE "exec/libraries.i"

INCLUDE "macros.i"
INCLUDE "devices/prtbase.i"

```

\*----- Imported Functions -----\*

```

XREF_EXE CloseLibrary
XREF_EXE OpenLibrary
XREF _AbsExecBase

XREF _PEDData
XREF _RenderBW
XREF _RenderColor

```

\*----- Exported Globals -----\*

```

XDEF _Init
XDEF _Expunge
XDEF _Open
XDEF _Close
XDEF _PD
XDEF _PED
XDEF _SysBase
XDEF _DOSBase
XDEF _GfxBase
XDEF _IntuitionBase

```

E - 26

\*\*\*\*\*

```

SECTION printer,DATA
_PD DC.L 0
_PED DC.L 0
_SysBase DC.L 0
_DOSBase DC.L 0
_GfxBase DC.L 0
_IntuitionBase DC.L 0

```

\*\*\*\*\*

```

SECTION printer,CODE
_Init:
MOVE.L 4(A7),_PD
LEA _PEDData(PC),A0
MOVE.L A0,_PED
MOVE.L A6,-(A7)
MOVE.L _AbsExecBase,A6
MOVE.L A6,_SysBase

```

```

* ;----- open the dos library
LEA DLName(PC),A1
MOVEQ #0,D0
CALLEXE OpenLibrary
MOVE.L D0,_DOSBase
BEQ initDLerr

```

```

* ;----- open the graphics library
LEA GLName(PC),A1
MOVEQ #0,D0
CALLEXE OpenLibrary
MOVE.L D0,_GfxBase
BEQ initGLerr

```

\* ;----- open the intuition library

```

LEA ILName(PC),A1
MOVEQ #0,D0
CALLEXE OpenLibrary
MOVE.L D0,_IntuitionBase
BEQ initILerr

```

```

pdIRts:
MOVEQ #0,D0
MOVE.L (A7)+,A6
RTS

```

```

initPAerr:
MOVE.L _IntuitionBase,A1
LINKEXE CloseLibrary

```

```

initILerr:
MOVE.L _GfxBase,A1
LINKEXE CloseLibrary

```

```

initGLerr:
MOVE.L _DOSBase,A1
LINKEXE CloseLibrary

```

```

initDLerr:
MOVEQ #-1,D0
BRA.S pdIRts

```

```

ILName:
DC.B 'intuition.library'
DC.B 0

```

```

DLName:
DC.B 'dos.library'
DC.B 0

```

```

GLName:
DC.B 'graphics.library'
DC.B 0
DS.W 0

```

\*-----

```

_Expunge:
MOVE.L _IntuitionBase,A1
LINKEXE CloseLibrary

```

```

MOVE.L _GfxBase,A1
LINKEXE CloseLibrary

```

```

MOVE.L _DOSBase,A1
LINKEXE CloseLibrary

```

```

*-----
_Open:
MOVE.L _PD,A0
CMPI.W #SHADE_COLOR,pd_Preferences+pf_PrintShade(A0)
BEQ.S colorRender
LEA _RenderBW,A0
MOVE.L A0,_PEDData+ped_Render

```

```

colorRender: BRA.S openEnd
 LEA _RenderColor,A0
 MOVE.L A0,_PEDData+ped_Render
openEnd: MOVEQ #0,D0
 RTS

```

```

*-----
_Close: MOVEQ #0,D0
 RTS
 END

```

okimate20/init.asm

```

TTL '$Header: init.asm,v 1.2 85/10/09 23:58:49 kodiak Exp $'

*
* Copyright 1985, Commodore-Amiga Inc. All rights reserved.
* No part of this program may be reproduced, transmitted,
* transcribed, stored in retrieval system, or translated into
* any language or computer language, in any form or by any
* means, electronic, mechanical, magnetic, optical, chemical,
* manual or otherwise, without the prior written permission of
* Commodore-Amiga Incorporated, 983 University Ave. Building #D,
* Los Gatos, California, 95030
*

*
* printer device functions
*
* Source Control
*-----
* $Header: init.asm,v 1.2 85/10/09 23:58:49 kodiak Exp $
*
* $Locker: $
*
* $Log: init.asm,v $
* Revision 1.2 85/10/09 23:58:49 kodiak
* replace reference to pdata w/ prtbase
*
* Revision 1.1 85/10/09 19:27:50 kodiak
* remove _stdout variable
*
* Revision 1.0 85/10/09 19:24:13 kodiak
* added to rcs for updating in version 1
*
* Revision 29.0 85/08/07 22:25:32 kodiak
* added to rcs for updating in version 29
*
* Revision 25.0 85/06/16 01:01:22 kodiak
* added to rcs
*

```

```

SECTION printer
*----- Included Files -----
INCLUDE "exec/types.i"
INCLUDE "exec/nodes.i"
INCLUDE "exec/lists.i"
INCLUDE "exec/memory.i"
INCLUDE "exec/ports.i"
INCLUDE "exec/libraries.i"

INCLUDE "macros.i"
INCLUDE "devices/prtbase.i"

```



\*----- Imported Functions -----\*

```

XREF_EXE CloseLibrary
XREF_EXE OpenLibrary
XREF _AbsExecBase

XREF _PEDData
XREF _RenderBW
XREF _RenderColor

```

\*----- Exported Globals -----\*

```

XDEF _Init
XDEF _Expunge
XDEF _Open
XDEF _Close
XDEF _PD
XDEF _PED
XDEF _SysBase
XDEF _DOSBase
XDEF _GfxBase
XDEF _IntuitionBase

```

E-28

\*\*\*\*\*

```

SECTION printer,DATA
_PD DC.L 0
_PED DC.L 0
_SysBase DC.L 0
_DOSBase DC.L 0
_GfxBase DC.L 0
_IntuitionBase DC.L 0

```

\*\*\*\*\*

```

SECTION printer,CODE
_Init:
MOVE.L 4(A7),_PD
LEA _PEDData(PC),A0
MOVE.L A0,_PED
MOVE.L A6,-(A7)
MOVE.L _AbsExecBase,A6
MOVE.L A6,_SysBase

```

```

* ;----- open the dos library
LEA DLName(PC),A1
MOVEQ #0,D0
CALLEXE OpenLibrary
MOVE.L D0,_DOSBase
BEQ initDLerr

```

```

* ;----- open the graphics library
LEA GLName(PC),A1
MOVEQ #0,D0
CALLEXE OpenLibrary
MOVE.L D0,_GfxBase
BEQ initGLErr

```

\* ;----- open the intuition library

```

LEA ILName(PC),A1
MOVEQ #0,D0
CALLEXE OpenLibrary
MOVE.L D0,_IntuitionBase
BEQ initILErr

```

```

pdIrts:
MOVEQ #0,D0
MOVE.L (A7)+,A6
RTS

```

```

initPAErr:
MOVE.L _IntuitionBase,A1
LINKEXE CloseLibrary

```

```

initILErr:
MOVE.L _GfxBase,A1
LINKEXE CloseLibrary

```

```

initGLErr:
MOVE.L _DOSBase,A1
LINKEXE CloseLibrary

```

```

initDLerr:
MOVEQ #-1,D0
BRA.S pdIrts

```

```

ILName:
DC.B 'intuition.library'
DC.B 0

```

```

DLName:
DC.B 'dos.library'
DC.B 0

```

```

GLName:
DC.B 'graphics.library'
DC.B 0
DS.W 0

```

\*-----

```

_Expunge:
MOVE.L _IntuitionBase,A1
LINKEXE CloseLibrary

```

```

MOVE.L _GfxBase,A1
LINKEXE CloseLibrary

```

```

MOVE.L _DOSBase,A1
LINKEXE CloseLibrary

```

\*-----

```

_Open:
MOVE.L _PD,A0
CMPI.W #SHADE_COLOR,pd_Preferences+pf_PrintShade(A0)
BEQ.S colorRender
LEA _RenderBW,A0
MOVE.L A0,_PEDData+ped_Render

```

```

colorRender: BRA.S openEnd
 LEA _RenderColor,A0
 MOVE.L A0,_PEDData+ped_Render

openEnd: MOVEQ #0,D0
 RTS

_close: MOVEQ #0,D0
 RTS
 END

```

Listing for okimate20/printertag.asm

```

TTL '$Header: printertag.asm,v 32.1 86/02/10 14:33:25 kodiak Exp $'

*
* Copyright 1985, Commodore-Amiga Inc. All rights reserved.
* No part of this program may be reproduced, transmitted,
* transcribed, stored in retrieval system, or translated into
* any language or computer language, in any form or by any
* means, electronic, mechanical, magnetic, optical, chemical,
* manual or otherwise, without the prior written permission of
* Commodore-Amiga Incorporated, 983 University Ave. Building #D,
* Los Gatos, California, 95030
*

*
* printer device dependent code tag
*
* Source Control
*
* $Header: printertag.asm,v 32.1 86/02/10 14:33:25 kodiak Exp $
* $Locker: $
* $Log: printertag.asm,v $
* Revision 32.1 86/02/10 14:33:25 kodiak
* add null 8BitChars field
*
* Revision 32.0 86/02/10 14:24:28 kodiak
* added to rcs for updating
*
* Revision 1.1 85/10/09 23:59:05 kodiak
* replace reference to pdata w/ prtbase
*
* Revision 1.0 85/10/09 23:58:58 kodiak
* added to rcs for updating in version 1
*
* Revision 29.1 85/07/31 18:27:25 kodiak
* change XDotInch from 144 to 120
*
* Revision 29.0 85/07/31 18:26:50 kodiak
* added to rcs for updating in version 29
*
* Revision 25.1 85/06/16 01:02:15 kodiak
* *** empty log message ***
*
* Revision 25.0 85/06/15 06:40:00 kodiak
* added to rcs
*
* Revision 25.0 85/06/13 18:53:36 kodiak
* added to rcs

SECTION printer
*----- Included Files -----
INCLUDE "exec/types.i"
INCLUDE "exec/nodes.i"

```

```

INCLUDE "exec/strings.i"

INCLUDE "devices/prtbase.i"

*----- Imported Names -----
XREF _Init
XREF _Expunge
XREF _Open
XREF _Close
XREF _CommandTable
XREF _PrinterSegmentData
XREF _DoSpecial

*----- Exported Names -----
XDEF _PEDData

MOVEQ #0,D0 ; show error for OpenLibrary()
RTS
DC.W VERSION
DC.W REVISION

_PEDData:
DC.L printerName
DC.L _Init
DC.L _Expunge
DC.L _Open
DC.L _Close
DC.B PPC_COLORGFX ; PrinterClass
DC.B PCC_YMC_BW ; ColorClass
DC.B 80 ; MaxColumns
DC.B 1 ; NumCharSets
DC.W 24 ; NumRows
DC.L 960 ; MaxXDots
DC.L 0 ; MaxYDots
DC.W 120 ; XDotsInch
DC.W 144 ; YDotsInch
DC.L _CommandTable ; Commands
DC.L _DoSpecial
DC.L 0 ; Render
DC.L 30
DC.L 0 ; 8BitChars

printerName:
STRING <'OKIMATE 20'>

END

```

Listing for okimate20/render.c

```

/*****
#include <exec/types.h>
#include <exec/nodes.h>
#include <exec/lists.h>
#include <exec/memory.h>
#include "devices/printer.h"
#include "devices/prtbase.h"

extern struct PrinterData *PD;
extern struct PrinterExtendedData *PED;

static UWORD rowsize;
static UWORD bufsize;
static UWORD bufptr;
static UWORD colors[4]; /* color ptrs */
static BYTE center; /* center picture flag */
static UBYTE bit_table[8] = {128, 64, 32, 16, 8, 4, 2, 1};
UBYTE *ptr;

/* for the OKIMATE 20 (Color) */
int RenderColor(ct, x, y, status) /* passed a color type */
UBYTE ct; /* the color type to use (0, 1, or 2) */
UWORD x, y; /* the x & y co-ordinates */
/* or the pc & pr print values, or special */
UBYTE status; /* print status (0-init, 1-enter pixel, 2-dump, 3-end) */
{
 UWORD i; /* mics. var */
 BYTE err; /* the error # */
 switch(status)
 {
 case 0: /*alloc memory for printer buffer */
 i = (center) ? ((PED->ped_MaxXDots - x) / 2 * 3) : 0;
 /* need this many more pixels */
 rowsize=(x*3+i); /* pc pixels per row x 3 colors on the OKIMATE 20 */
 bufsize=(rowsize*3+31); /* buffer size required for OKIMATE 20 */
 colors[0] = 0 + i;
 colors[1] = 7 + i;
 colors[2] = 9+rowsize+7 + i;
 colors[3] = (9+rowsize)*2+7 + i;
 PD->pd_PrintBuf = (UBYTE *)
 AllocMem(bufsize*2, MEMF_PUBLIC); /* alloc public mem */
 if (err=(PD->pd_PrintBuf == 0)) return(err);
 bufptr=0;
 /* set line spacing to 24 printer lines (24/144 -> 36/216 inch) */
 return((*PD->pd_PWrite)("\033\044", 3)); /* thats Esc3<36 */
 break;

 case 1: /* put pixel in buffer (called 69,120 times/print cycle) */
 i = bufptr+(y % 24)/8 + x*3 + colors[ct];
 /* calc which byte to use */
 PD->pd_PrintBuf[i] = PD->pd_PrintBuf[i] | (1 << (7-(y&7)));
 /* set pixel */
 PD->pd_PrintBuf[bufptr + ((y&24)>>3) + (x<<1) + x + colors[ct]]
 |= bit_table[y & 7]; /* calc byte posn and set pixel */
 return(0); /* flag all ok */
 break;
 }
}

```

```

case 2 : /* dump buffer to printer */
 if (err=(*(PD->pd_PWrite))(&(PD->pd_PrintBuf[bufptr]),
 bufsize)) return(err);
 bufptr = bufsize - bufptr;
 return(0); /* flag all ok */
 break;

case 3 : /* clear and init buffer (called once/print cycle) */
 ptr = &PD->pd_PrintBuf[bufptr];
 ptr++ = 27; / (bufptr) */
 ptr++ = 25; / align ribbon (bufptr+1)*/
 i = bufsize - 2; /* less the previous cmds */
 while (i-- > 0) *ptr++ = 0; /* clear buffer (executed 8,571 - 2 times) */
 for (ct=0; ct<3; ct++) { /* for all color types */
 PD->pd_PrintBuf[2+bufptr+ct*(rowsize+9)] = 27;
 PD->pd_PrintBuf[3+bufptr+ct*(rowsize+9)] = '%';
 PD->pd_PrintBuf[4+bufptr+ct*(rowsize+9)] = '0';
 /* enter 24-dot mode */
 PD->pd_PrintBuf[5+bufptr+ct*(rowsize+9)] = (rowsize/3) & 0xff;
 PD->pd_PrintBuf[6+bufptr+ct*(rowsize+9)] = (rowsize/3) >> 8;
 /* set # of dots */
 PD->pd_PrintBuf[1+bufptr+(ct+1)*(rowsize+9)] = 13;
 /* advance color */
 }
 PD->pd_PrintBuf[bufptr+bufsize-2] = 10; /* lf */
 PD->pd_PrintBuf[bufptr+bufsize-1] = 13; /* cr */
 return(0); /* flag all ok */
 break;

case 4 : /* free the print buffer memory */
 err=(*(PD->pd_PBothReady)); /* wait for both buffers to empty */
 FreeMem(PD->pd_PrintBuf,bufsize*2); /* free printers memory */
 return(err); /* return status */
 break;

case 5 : /* io_special flag call */
 center = x & SPECIAL_CENTER; /* set center flag */
 return(0); /* flag all ok */
 break;

default: return(0); /* flag all ok */
}
}
/*****
*/
/* for the OKIMATE 20 (b/w) */
int RenderBW(ct, x, y, status) /* passed a color type */
 UBYTE ct; /* not used with b/w printers */
 UWORD x, y; /* the x & y co-ordinates */
 /* or the pc & pr print values, or special */
 UBYTE status /* print status (0-init, 1-enter pixel,
 * 2-dump, 3-end) */
{
 UWORD i; /* mics. var */
 BYTE err; /* the error # */
 static UWORD offset;

 switch(status)
 {
 case 0 : /*alloc memory for printer buffer */
 i = (center) ? ((PD->ped_MaxXDots - x) / 2 * 3) : 0;
 /* need this many more pixels */
 offset = 5 + i;
 rowsize=(x*3+i); /* pc pixels per row x 3 blocks on the OKIMATE 20 bw */
 bufsize=(rowsize+7); /* buffer size required for OKIMATE 20 bw */
 PD->pd_PrintBuf = (UBYTE *)
 AllocMem(bufsize*2,MEMF_PUBLIC); /* alloc public mem */
 if (err=(PD->pd_PWrite) == 0) return(err);
 bufptr = 0; /* init to first buffer */
 /* set line spacing to 24 printer lines (24/144 -> 36/216 inch) */
 return(*(PD->pd_PWrite))("\033\044", 3); /* thats Esc<36> */
 break;

 case 1 : /* put pixel in buffer */
 i = bufptr + (y % 24)/8 + x*3 + 5; /* calc which byte to use */
 PD->pd_PrintBuf[i] = PD->pd_PrintBuf[i] | (1 << (7-(y&7)));
 /* fill print buffer */
 PD->pd_PrintBuf[bufptr + (y % 24)>>3] + (x<<1) + x + offset]
 |= bit_table[y&7];
 return(0); /* flag all ok */
 break;

 case 2 : /* dump buffer to printer */
 if (err=(*(PD->pd_PWrite))(&(PD->pd_PrintBuf[bufptr]),
 bufsize)) return(err);
 bufptr = bufsize - bufptr; /* switch to other buffer */
 return(0); /* flag all ok */
 break;

 case 3 : /* clear and init buffer */
 for (i=bufptr; i<bufptr+bufsize; i++)
 PD->pd_PrintBuf[i] = 0; /* clear buffer */
 ptr = &PD->pd_PrintBuf[bufptr];
 i = bufsize;
 while (i-- > 0) *ptr++ = 0; /* clear buffer */
 PD->pd_PrintBuf[bufptr] = 27;
 PD->pd_PrintBuf[bufptr+1] = '%';
 PD->pd_PrintBuf[bufptr+2] = '0'; /* enter 24-dot mode */
 PD->pd_PrintBuf[bufptr+3] = (rowsize/3) & 0xff;
 PD->pd_PrintBuf[bufptr+4] = (rowsize/3) >> 8;
 /* there is rowsize dots */
 PD->pd_PrintBuf[bufptr+bufsize-2] = 13; /* cr */
 PD->pd_PrintBuf[bufptr+bufsize-1] = 10; /* lf */
 return(0); /* flag all ok */
 break;

 case 4 : /* free the print buffer memory */
 err=(*(PD->pd_PBothReady)); /* wait for both buffers to empty */
 FreeMem(PD->pd_PrintBuf,bufsize*2); /* free the print buffer mem */
 return(err);
 break;

 case 5 : /* io special flag call */
 center = x & SPECIAL_CENTER; /* set center flag */
 return(0); /* flag all ok */
 break;

 default: return(0);
 }
}

```

**Amiga  
Printer Support  
Information**

## General Information

The Amiga printer drivers are among the most complete in the industry. We have made every effort to provide support for a wide variety of printers and an extensive list of features. The Preferences tool on your Workbench disk lists the available printers that are supported. (The default printer settings in Preferences are for the Epson printers.) See *Introduction to Amiga* for instructions on changing the Preferences settings.

This document provides the following information:

How to use the Preferences printer settings with the printer device

How to use the parallel and serial devices

How to use the printer.library routines for direct printer I/O

How to set the standard cables and switch settings for printers

For an unsupported printer, use the "Custom/Generic" Preferences setting. See the *Amiga ROM Kernal Manual* for instructions on constructing a custom printer driver for an unsupported printer.

AmigaDos provides three "handlers," or interface routines, for printer I/O:

PAR: parallel device

SER: serial device

PRT: printer device

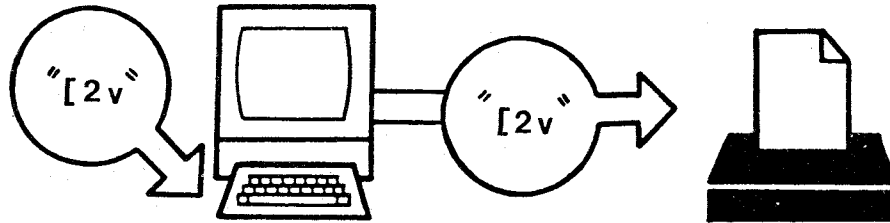
Each of these handlers translates the device-independent file system calls, such as Write() and Open(), into the appropriate message traffic to the printer devices that are implemented in Exec. Exec is the multi-tasking kernel of the Amiga.

The "PAR:" handler uses the "parallel.device", which is the Exec code that manages the parallel port connector on the back of your Amiga. Similarly, the "SER:" handler uses the device "serial.device" to manage the serial port connector. Note that, aside from the baud rate setting for the serial port, the Preferences printer settings have no effect on the function of the PAR: and SER: handlers. The characters sent to the printer using these devices are not examined or converted.

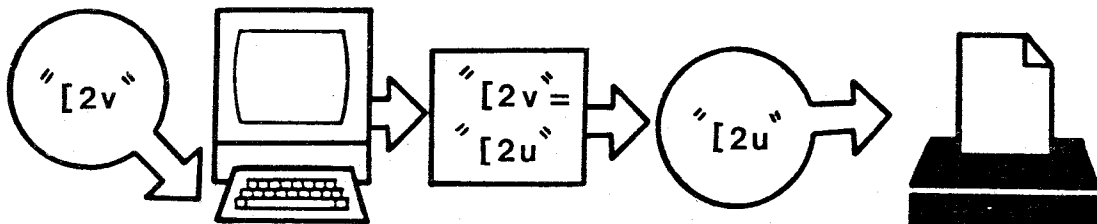
In other words, when you send output to PAR: or SER:, your application is talking straight through to the hardware with no intervening levels of interpretation. If you have a printer connected to your parallel port, escape sequences sent to PAR: will reach it directly and will have whatever effect they are defined to have by the printer manufacturer.

On the other hand, the PRT: handler uses the Exec device, "printer.device." The printer device uses the information it finds in the current Preferences settings to understand which kind of printer you have connected and how you want it to be used. The printer device can talk to either the parallel or the serial device, depending on the current Preferences setting.

The following figures illustrate the difference between sending a particular escape sequence to a printer using the PRT: handler instead of the PAR: or SER: handlers.



*Figure 1: Printer I/O Through SER: or PAR: Handlers*



*Figure 2: Printer I/O Through PRT: Handler Via Preferences Tool*

The escape sequence for turning on superscripts is defined for the Epson JX-80 to be the escape character (ASCII code 27) plus the string, "[2u". However, the Amiga printer-independent escape sequence for a superscript is "[2v". Therefore, the printer driver for this particular printer must convert the latter string into the former in order for the printer to effect superscript mode. The PAR: and SER: handlers perform no such conversion.

Deciding which printer handler to use depends on the nature of your application. If you use the printer device (PRT:), you can write code that is largely independent of the type of printer your customers have attached to their Amigas. This is the recommended method.

Printing to PAR: or SER: is fairly straightforward. Keep in mind that a standard AmigaDOS text file uses LF (line feed) as a line separator--not CR or CR-LF) and that a file may or may not have an LF at the end. You may wish to add a carriage return character to the ends of your lines of text. Or, if your printer offers the option, you can flip the switch that automatically gives a CR when the printer receives an LF.

The CLI commands expect you to use the handler names as file parameters. For example, you can send a file to the printer with the command,

`copy myfile to prt:`

If you want to send output to the printer using the AmigaDOS file system routines directly, you must Open() one of the handlers and do Write() calls to it.

Similarly, you should use the handler names with I/O to the printer from languages such as ABasiC. Note that--for compatibility--Microsoft's Amiga Basic defines LPT1: to be the same as PRT:.

You can circumvent the handlers entirely and perform a direct OpenDevice() on the Exec device of interest to you. You then pass I/O request blocks to the device using the I/O calls provided by Exec (such as DoIO()). Doing so provides greater flexibility, such as allowing asynchronous I/O and setting device parameters (serial baud rate, for example). By using the printer.library, you have full control over the printer.

Note that you must open the printer.library directly in order to use the command names instead of the defined escape sequences. See Table 3 for a list of the printer features and their command names. See the *Amiga ROM Kernal Manual* for more information on calling system library and device routines.

Note the following information regarding sending I/O between the Amiga and various printers:

### Printer Device (PRT:)

The printer device understands only its own, *printer-independent*, escape sequences. It converts these escape sequences into the printer-specific escape sequences appropriate for the printer currently selected in Preferences. In addition, the Initialize function (which is invoked when you open the printer device or when you send it the Initialize escape sequence) causes the appropriate escapes to be sent to your printer to configure it according to the options you have selected in Preferences. This, for example, is how your margin settings are sent to the printer.

Note that, when you use the printer device, you should *turn off* any option on your printer that provides an automatic CR, LF, or CR-LF whenever the printer receives an LF. The printer device provides end of line CR-LFs as needed.

Also keep in mind that--in addition to the alphanumeric printing described here--the printer device provides for black and white, grey-scale, and full color raster-graphics printing. This function is only available when your application talks *directly* to the printer device and *not* through the AmigaDOS PRT: handler. See the *Amiga ROM Kernal Manual* for an example.

### Serial and Parallel Handlers (SER:, PAR:)

The Preferences tool printer settings have no effect on the function of the PAR: and SER: handlers (other than setting the baud rate used by SER:, as noted above). Any special function you want your printer to perform is up to you. You must choose the correct escape sequences to send, including even initialization functions such as the setting of margins. Clearly, you must know which printer is connected to your Amiga and whether it is connected to the serial or the parallel port. This is not the recommended method of controlling printers.

Specific serial device features (for SER:) that you *cannot* set in Preferences include:



|                                                    |                        |
|----------------------------------------------------|------------------------|
| Hardware (7-wire) or software (3-wire) handshaking | (XON XOFF always used) |
| Number of bits                                     | (8 bits always used)   |
| Parity                                             | (none)                 |

See the *ROM Kernel Manual* for details on setting these features.

## PRINTER.LIBRARY

With the printer.library, you not only can send escape sequences to the printer, you can also call the printer-unique entry point, "PRT". This entry point allows you to control the printer directly--the necessary escape sequences will be generated for you.

In addition, there is a printer-unique function, "RAW\_WRITE" that sends characters without converting them. This functions the same as SER: and PAR:, except that you don't need to know which port is connected to the printer.

## Types of Supported Printers

The available printers that are supported for the Amiga include both whole character (daisy wheel) and dot matrix (wire, ink jet, and laser) types. As with printer capabilities, printer prices range widely, from just over \$200 to over \$3500. In general, the dot matrix printers are capable of graphics output, while "whole character" printers are not.

Every attempt has been made to support a given feature on each printer that, itself, supports that feature. For example, the daisy wheel printers lack the capability to produce characters such as enlarged or italic print. Similarly, the dot matrix printers often lack such features as proportional spacing.

None of the supported printers currently supports all of the available features. (The Epson JX-80 and the HP LaserJet come closest.) Whenever the system requests an unsupported feature, the PRT: handler simply ignores that request. (The "generic" printer driver currently ignores all feature requests.)

If two or more features are each available for a particular printer, they should be usable in combination. For example, Bold-Italic-Underscore is a possible style for many printers.

If your printer is not among those supported for the Amiga, you have two options. If your printer shares a number of common features with one of the supported printers, you can select that printer in Preferences.

Keep in mind, however, that one or more of the chosen printer's features might not produce a similar effect on your printer.

Your second option is to select "Custom" from the list of supported printers in Preferences and "Generic" as the custom printer name. You can then construct a custom printer driver following the directions in the *Amiga ROM Kernel Manual*.

The following table lists the printers that are currently supported for the Amiga, grouped according to print technology.

**Table 1: Printers Supported on the Amiga**

**Dot Matrix (Wire), Parallel**

| <b>Manufacturer</b> | <b>Model</b>            |
|---------------------|-------------------------|
| Commodore           | CBM MPS 1000            |
| Epson               | Epson JX-80             |
| Epson               | Epson MX-80, FX-80, ... |
| Okimate             | Okimate 20              |

**Daisy Wheel, Parallel**

| <b>Manufacturer</b> | <b>Model</b>                 |
|---------------------|------------------------------|
| Alphacom            | Alphapro 101                 |
| Brother             | HR-15XL                      |
| Diablo              | 630 (Some models are serial) |
| Diablo              | Advantage D25                |
| Qume                | LetterPro 20                 |

**Ink Jet, Parallel**

| <b>Manufacturer</b> | <b>Model</b> |
|---------------------|--------------|
| Diablo              | C-150        |

**Laser, Serial**

| <b>Manufacturer</b> | <b>Model</b>   |
|---------------------|----------------|
| Hewlett Packard     | Laser Jet      |
| Hewlett Packard     | Laser Jet Plus |

**Other (Custom)**

Limited support is offered for a "generic" printer.

**Table 2: Printer Features Supported on the Amiga**

Legend:

-----

ISO indicates that the sequence has been defined by the International Standards Organization. This is also very similar to ANSI x3.64.

DEC indicates a control sequence defined by Digital Equipment Corporation.

- \* Entire escape sequence consists of ESC (ASCII 27) plus indicated code.
- \*\* Near Letter Quality
- \*\*\* Sequence unique to Amiga
- † Paper perforation skip, *n* lines

| Code* | Description                                   | Defined | Alphacom<br>AlphaPro101 | Brother<br>HR-15XL | Diablo630 | CBM<br>MPS-1000 | Epson<br>JX-80 | Epson<br>X-80 | Diablo<br>AdvantageD25 | Diablo<br>C-150 | Okimate<br>20 | HP<br>LaserJet | HP<br>LaserJet Plus | Qume<br>LetterPro 20 |
|-------|-----------------------------------------------|---------|-------------------------|--------------------|-----------|-----------------|----------------|---------------|------------------------|-----------------|---------------|----------------|---------------------|----------------------|
| c     | Reset                                         | ISO     | x                       | x                  | x         | x               | x              | x             | x                      | x               | x             | x              | x                   | x                    |
| #1    | Initialize                                    | ***     | x                       | x                  | x         | x               | x              | x             | x                      | x               | x             | x              | x                   | x                    |
| D     | Line feed                                     | ISO     | x                       | x                  | x         | x               | x              | x             | x                      | x               |               | x              | x                   | x                    |
| E     | Line feed, CR                                 | ISO     | x                       | x                  | x         | x               | x              | x             | x                      | x               | x             | x              | x                   | x                    |
| M     | Reverse line feed                             | ISO     | x                       | x                  | x         |                 | x              |               | x                      |                 |               | x              | x                   | x                    |
| [0m   | Normal char. set                              | ISO     | x                       | x                  | x         | x               | x              | x             | x                      |                 | x             | x              | x                   | x                    |
| [3m   | Italics on                                    | ISO     |                         |                    |           |                 | x              | x             |                        |                 | x             | x              | x                   |                      |
| [23m  | Italics off                                   | ISO     |                         |                    |           |                 | x              | x             |                        |                 | x             | x              | x                   |                      |
| [4m   | Underline on                                  | ISO     | x                       | x                  | x         | x               | x              | x             | x                      |                 | x             | x              | x                   | x                    |
| [24m  | Underline off                                 | ISO     | x                       | x                  | x         | x               | x              | x             | x                      |                 | x             | x              | x                   | x                    |
| [1m   | Boldface on                                   | ISO     | x                       | x                  | x         | x               | x              | x             | x                      |                 |               | x              | x                   | x                    |
| [22m  | Boldface off                                  | ISO     | x                       | x                  | x         | x               | x              | x             | x                      |                 |               | x              | x                   | x                    |
| [nm   | Set foreground color<br>( <i>n</i> = {30-39}) | ISO     | x                       | x                  | x         |                 | x              |               | x                      | x               |               |                |                     |                      |
| [nm   | Set background color<br>( <i>n</i> = {40-49}) | ISO     |                         |                    |           |                 |                |               |                        | x               |               |                |                     |                      |
| [0w   | Normal pitch                                  | DEC     | x                       | x                  | x         | x               | x              | x             | x                      |                 | x             | x              | x                   | x                    |
| [2w   | Elite on                                      | DEC     | x                       | x                  | x         | x               | x              | x             | x                      |                 | x             | x              | x                   | x                    |
| [1w   | Elite off                                     | DEC     | x                       | x                  | x         | x               | x              | x             | x                      |                 | x             | x              | x                   | x                    |
| [4w   | Condensed fine on                             | DEC     | x                       | x                  | x         | x               | x              | x             | x                      |                 | x             | x              | x                   | x                    |
| [3w   | Condensed off                                 | DEC     | x                       | x                  | x         | x               | x              | x             | x                      |                 | x             | x              | x                   | x                    |
| [6w   | Enlarged on                                   | DEC     |                         |                    |           | x               | x              | x             |                        |                 | x             |                |                     |                      |
| [5w   | Enlarged off                                  | DEC     |                         |                    |           | x               | x              | x             |                        |                 | x             |                |                     |                      |
| [6"z  | Shadow print on                               | DEC     | x                       | x                  | x         |                 |                |               | x                      |                 |               | x              | x                   | x                    |
| [5"z  | Shadow print off                              | DEC     | x                       | x                  | x         |                 |                |               | x                      |                 |               | x              | x                   | x                    |
| [4"z  | Doublestrike on                               | DEC     | x                       | x                  | x         | x               | x              | x             | x                      |                 |               | x              | x                   | x                    |
| [3"z  | Doublestrike off                              | DEC     | x                       | x                  | x         | x               | x              | x             | x                      |                 |               | x              | x                   | x                    |
| [2"z  | NLQ on **                                     | DEC     |                         |                    |           | x               | x              | x             |                        |                 | x             |                |                     |                      |
| [1"z  | NLQ off **                                    | DEC     |                         |                    |           | x               | x              | x             |                        |                 | x             |                |                     |                      |

| Code | Description             | Defined | Alphacom<br>AlphaPro101 | Brother<br>HR-15XL | Diablo630 | CBM<br>MPS-1000 | Epson<br>JX-80 | Epson<br>X-80 | Diablo<br>AdvantageD25 | Diablo<br>C-150 | Okimate<br>20 | HP<br>LaserJet | HP<br>LaserJet Plus | Qume<br>LetterPro 20 |
|------|-------------------------|---------|-------------------------|--------------------|-----------|-----------------|----------------|---------------|------------------------|-----------------|---------------|----------------|---------------------|----------------------|
| [2v  | Superscript on          | ***     | x                       | x                  | x         | x               | x              | x             | x                      |                 | x             | x              | x                   | x                    |
| [1v  | Superscript off         | ***     | x                       | x                  | x         | x               | x              | x             | x                      |                 | x             | x              | x                   | x                    |
| [4v  | Subscript on            | ***     | x                       | x                  | x         | x               | x              | x             | x                      |                 | x             | x              | x                   | x                    |
| [3v  | Subscript off           | ***     | x                       | x                  | x         | x               | x              | x             | x                      |                 | x             | x              | x                   | x                    |
| [0v  | Normalize the line      | ***     | x                       | x                  | x         | x               | x              | x             | x                      |                 | x             | x              | x                   | x                    |
| L    | Partial line up         | ISO     | x                       | x                  | x         | x               | x              | x             | x                      |                 | x             | x              | x                   | x                    |
| K    | Partial line down       | ISO     | x                       | x                  | x         | x               | x              | x             | x                      |                 | x             | x              | x                   | x                    |
| (B   | U.S. char. set          | DEC     |                         |                    |           |                 | x              | x             |                        |                 |               | x              | x                   |                      |
| (R   | French " "              | DEC     |                         |                    |           |                 | x              | x             |                        |                 |               | x              | x                   |                      |
| (K   | German " "              | DEC     |                         |                    |           |                 | x              | x             |                        |                 |               | x              | x                   |                      |
| (A   | UK " "                  | DEC     |                         |                    |           |                 | x              | x             |                        |                 |               | x              | x                   |                      |
| (E   | Danish I " "            | DEC     |                         |                    |           |                 | x              | x             |                        |                 |               | x              | x                   |                      |
| (H   | Swedish " "             | DEC     |                         |                    |           |                 | x              | x             |                        |                 |               | x              | x                   |                      |
| (Y   | Italian " "             | DEC     |                         |                    |           |                 | x              | x             |                        |                 |               | x              | x                   |                      |
| (Z   | Spanish " "             | DEC     |                         |                    |           |                 | x              | x             |                        |                 |               | x              | x                   |                      |
| (J   | Japanese " "            | ***     |                         |                    |           |                 | x              | x             |                        |                 |               | x              | x                   |                      |
| (6   | Norwegian " "           | DEC     |                         |                    |           |                 | x              | x             |                        |                 |               | x              | x                   |                      |
| (C   | Danish II " "           | ***     |                         |                    |           |                 | x              | x             |                        |                 |               | x              | x                   |                      |
| [2p  | Proportional on         | ***     |                         | x                  | x         | x               | x              | x             | x                      |                 |               | x              | x                   | x                    |
| [1p  | Proportional off        | ***     |                         | x                  | x         | x               | x              | x             | x                      |                 |               | x              | x                   | x                    |
| [0p  | Proportional clear      | ***     |                         | x                  | x         |                 |                |               | x                      |                 |               | x              | x                   | x                    |
| [n E | Set prop. offset (n)    | ISO     |                         |                    |           |                 |                |               |                        |                 |               |                |                     |                      |
| [5 F | Auto left justify       | ISO     |                         |                    | x         |                 |                | x             |                        |                 |               |                |                     |                      |
| [7 F | Auto right justify      | ISO     |                         |                    |           |                 |                | x             |                        |                 |               |                |                     |                      |
| [6 F | Auto full justify       | ISO     |                         |                    |           |                 |                | x             |                        |                 |               |                |                     |                      |
| [0 F | Justify off             | ISO     |                         |                    | x         |                 |                | x             | x                      |                 |               |                |                     |                      |
| [3 F | Letter space (justify)  | ISO     |                         |                    |           |                 |                | x             |                        |                 |               |                |                     |                      |
| [1 F | Word fill (auto center) | ISO     |                         |                    | x         |                 |                | x             |                        |                 |               |                |                     |                      |
| [0z  | 1/8" line spacing       | ***     | x                       | x                  | x         | x               | x              | x             | x                      |                 | x             | x              | x                   | x                    |
| [1z  | 1/6" line spacing       | ***     | x                       | x                  | x         | x               | x              | x             | x                      |                 | x             | x              | x                   | x                    |
| [nt  | Set form length (n)     | DEC     | x                       | x                  | x         | x               | x              | x             | x                      | x               | x             | x              | x                   | x                    |
| [nq  | Perf skip (n>0) †       | ***     |                         |                    |           | x               | x              | x             |                        |                 | x             | x              | x                   |                      |
| [0q  | Perf skip off           | ***     |                         |                    |           | x               | x              | x             |                        |                 | x             | x              | x                   |                      |

| Code    | Description          | Defined | Alphacom<br>AlphaPro101 | Brother<br>HR-15XL | Diablo630 | CBM<br>MPS-1000 | Epson<br>JX-80 | Epson<br>X-80 | Diablo<br>AdvantageD25 | Diablo<br>C-150 | Okimate<br>20 | HP<br>LaserJet | HP<br>LaserJet Plus | Quime<br>LetterPro 20 |
|---------|----------------------|---------|-------------------------|--------------------|-----------|-----------------|----------------|---------------|------------------------|-----------------|---------------|----------------|---------------------|-----------------------|
| #9      | Left margin set      | ***     | x                       | x                  | x         |                 |                |               | x                      | x               |               |                |                     | x                     |
| #0      | Right margin set     | ***     | x                       | x                  | x         |                 |                |               | x                      | x               |               |                |                     | x                     |
| #8      | Top margin set       | ***     | x                       | x                  | x         |                 |                |               | x                      |                 |               |                |                     |                       |
| #2      | Bottom margin set    | ***     | x                       | x                  | x         |                 |                |               | x                      |                 |               |                |                     |                       |
| [n1;n2r | Top;Bottom margins   | DEC     |                         |                    |           |                 |                |               |                        |                 |               | x              | x                   |                       |
| [n1;n2s | Left;Right margins   | DEC     | x                       | x                  | x         | x               | x              | x             | x                      | x               |               | x              | x                   | x                     |
| #3      | Clear margins        | ***     | x                       | x                  | x         | x               | x              | x             | x                      | x               |               | x              | x                   | x                     |
| H       | Set horiz. tab       | ISO     |                         | x                  | x         |                 |                |               | x                      | x               |               |                |                     |                       |
| J       | Set vert. tab        | ISO     |                         | x                  | x         |                 |                |               | x                      |                 |               |                |                     |                       |
| [0g     | Clear horiz. tab     | ISO     |                         | x                  | x         |                 |                |               | x                      | x               |               |                |                     |                       |
| [3g     | Clear all hor. tabs  | ISO     |                         | x                  | x         | x               | x              | x             | x                      | x               | x             |                |                     |                       |
| [1g     | Clear vert. tab      | ISO     |                         |                    |           |                 |                |               |                        |                 | x             |                |                     |                       |
| [4g     | Clear all vert. tabs | ISO     |                         | x                  |           | x               | x              | x             |                        |                 |               |                |                     |                       |
| #4      | Clear all h & v tabs | ***     |                         | x                  | x         | x               | x              | x             | x                      | x               |               |                |                     |                       |
| #5      | Set default tabs     | ***     |                         | x                  | x         | x               | x              | x             | x                      | x               | x             |                |                     |                       |
| [n"x    | (Extended commands)  | ***     |                         |                    |           |                 |                |               |                        |                 |               |                |                     |                       |

**Table 3: Printer Command Definitions**

The following table describes the supported printer functions. You can use the escape sequences with PRT: and the printer.library. To use the command names, open the printer.library directly.

Again, recall that SER: and PAR: will ignore all of these and pass them directly on to the attached device.

| Cmd Name | Escape Sequence     | Function             | Defined by:*  |
|----------|---------------------|----------------------|---------------|
| aRIS     | ESCc                | reset                | ISO           |
| aRIN     | ESC#1               | initialize           | ***           |
| aIND     | ESCD                | lf                   | ISO           |
| aNEL     | ESCE                | return,lf            | ISO           |
| aRI      | ESCM                | reverse lf           | ISO           |
| aSGR0    | ESC[0m              | normal char set      | ISO           |
| aSGR3    | ESC[3m              | italics on           | ISO           |
| aSGR23   | ESC[23m             | italics off          | ISO           |
| aSGR4    | ESC[4m              | underline on         | ISO           |
| aSGR24   | ESC[24m             | underline off        | ISO           |
| aSGR1    | ESC[1m              | boldface on          | ISO           |
| aSGR22   | ESC[22m             | boldface off         | ISO           |
| aSFC     | ESC[nm (n= {30-39}) | set foreground color | ISO           |
| aSBC     | ESC[nm (n= {40-49}) | set background color | ISO           |
| aSHORP0  | ESC[0w              | normal pitch         | DEC           |
| aSHORP2  | ESC[2w              | elite on             | DEC           |
| aSHORP1  | ESC[1w              | elite off            | DEC           |
| aSHORP4  | ESC[4w              | condensed fine on    | DEC           |
| aSHORP3  | ESC[3w              | condensed off        | DEC           |
| aSHORP6  | ESC[6w              | enlarged on          | DEC           |
| aSHORP5  | ESC[5w              | enlarged off         | DEC           |
| aDEN6    | ESC[6"z             | shadow print on      | DEC (sort of) |
| aDEN5    | ESC[5"z             | shadow print off     | DEC           |
| aDEN4    | ESC[4"z             | doublestrike on      | DEC           |
| aDEN3    | ESC[3"z             | doublestrike off     | DEC           |
| aDEN2    | ESC[2"z             | NLQ on               | DEC           |
| aDEN1    | ESC[1"z             | NLQ off              | DEC           |
| aSUS2    | ESC[2v              | superscript on       | ***           |
| aSUS1    | ESC[1v              | superscript off      | ***           |
| aSUS4    | ESC[4v              | subscript on         | ***           |
| aSUS3    | ESC[3v              | subscript off        | ***           |
| aSUS0    | ESC[0v              | normalize the line   | ***           |
| aPLU     | ESCL                | partial line up      | ISO           |
| aPLD     | ESCK                | partial line down    | ISO           |
| aFNT0    | ESC(B               | US char set          | DEC           |
| aFNT1    | ESC(R               | French char set      | DEC           |
| aFNT2    | ESC(K               | German char set      | DEC           |

|         |            |                         |               |
|---------|------------|-------------------------|---------------|
| aFNT3   | ESC(A      | UK char set             | DEC           |
| aFNT4   | ESC(E      | Danish I char set       | DEC           |
| aFNT5   | ESC(H      | Swedish char set        | DEC           |
| aFNT6   | ESC(Y      | Italian char set        | DEC           |
| aFNT7   | ESC(Z      | Spanish char set        | DEC           |
| aFNT8   | ESC(J      | Japanese char set       | ***           |
| aFNT9   | ESC(6      | Norweign char set       | DEC           |
| aFNT10  | ESC(C      | Danish II char set      | ***           |
| aPROP2  | ESC[2p     | proportional on         | ***           |
| aPROP1  | ESC[1p     | proportional off        | ***           |
| aPROP0  | ESC[0p     | proportional clear      | ***           |
| aTSS    | ESC[n E    | set proportional offset | ISO           |
| aJFY5   | ESC[5 F    | auto left justify       | ISO           |
| aJFY7   | ESC[7 F    | auto right justify      | ISO           |
| aJFY6   | ESC[6 F    | auto full justify       | ISO           |
| aJFY0   | ESC[0 F    | auto justify off        | ISO           |
| aJFY3   | ESC[3 F    | letter space (justify)  | ISO (special) |
| aJFY1   | ESC[1 F    | word fill(auto center)  | ISO (special) |
| aVERP0  | ESC[0z     | 1/8" line spacing       | ***           |
| aVERP1  | ESC[1z     | 1/6" line spacing       | ***           |
| aSLPP   | ESC[nt     | set form length n       | DEC           |
| aPERF   | ESC[nq     | perf skip n (n>0)       | ***           |
| aPERF0  | ESC[0q     | perf skip off           | ***           |
| aLMS    | ESC#9      | Left margin set         | ***           |
| aRMS    | ESC#0      | Right margin set        | ***           |
| aTMS    | ESC#8      | Top margin set          | ***           |
| aBMS    | ESC#2      | Bottom marg set         | ***           |
| aSTBM   | ESC[n1;n2r | T&B margins             | DEC           |
| aSLRM   | ESC[n1;n2s | L&R margin              | DEC           |
| aCAM    | ESC#3      | Clear margins           | ***           |
| aHTS    | ESCH       | Set horiz tab           | ISO           |
| aVTS    | ESCJ       | Set vertical tabs       | ISO           |
| aTBC0   | ESC[0g     | Clr horiz tab           | ISO           |
| aTBC3   | ESC[3g     | Clear all h tab         | ISO           |
| aTBC1   | ESC[1g     | Clr vertical tabs       | ISO           |
| aTBC4   | ESC[4g     | Clr all v tabs          | ISO           |
| aTBCALL | ESC#4      | Clr all h & v tabs      | ***           |
| aTBSALL | ESC#5      | Set default tabs        | ***           |
| aEXTEND | ESC[n"x    | extended commands       | ***           |

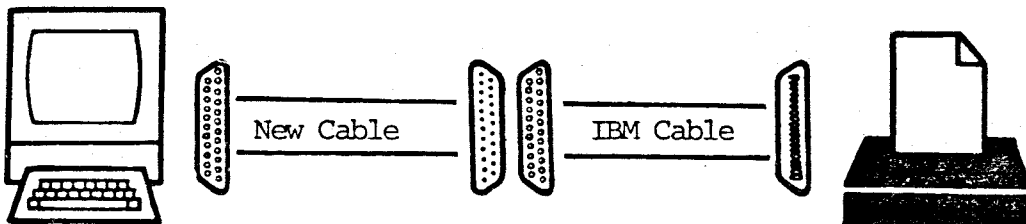
\* See legend for Table 2.

## Standard Cable Connections for Printers

If you want to connect a printer to the Amiga parallel port (25 pin female) and you have an IBM PC parallel to Centronics (36 pin) cable, make the following 25 pin female to female cable:

| Amiga Side         | IBM Cable Side |
|--------------------|----------------|
| 1-13               | 1-13           |
| 14-16 (cut)        |                |
| 17-22              | 17-22          |
| 23 (cut)           |                |
| 24                 | 24             |
| 26 connect over to | 16             |

Now arrange as follows:



Note: Don't connect pin 14 (parallel); it causes extra line feeds on Epson printers.

### Amiga to Centronics Adapter

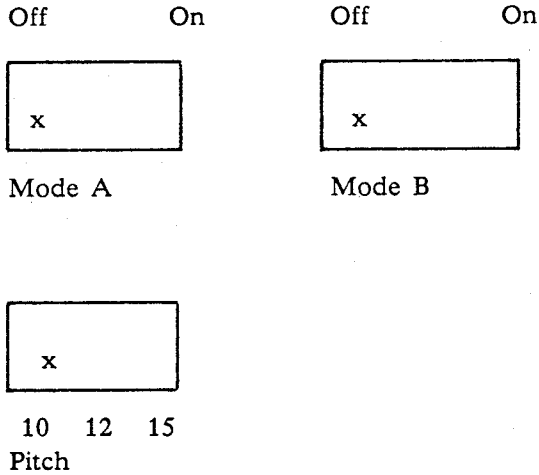
| Amiga Side         | Centronics Side |
|--------------------|-----------------|
| 1-13 (straight)    | 1-13            |
| 14-16 (cut)        |                 |
| 17-22 (straight)   | 17-22           |
| 23 (cut)           |                 |
| 24                 | 24              |
| 25 connect over to | 16              |
|                    | 25 (cut)        |



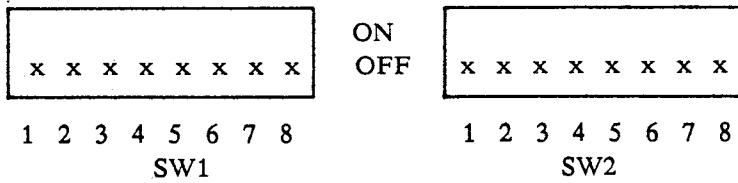
### Table 4: Standard Switch Settings for Printers

The standard switch settings for the Amiga supported printers are as follows:

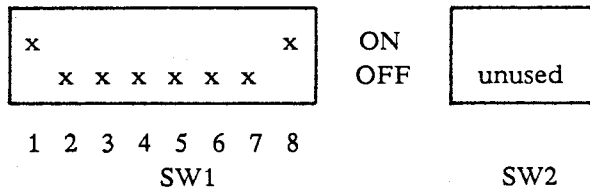
#### Alphacom AlphaPro 101



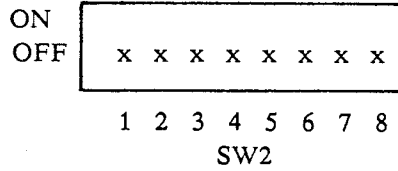
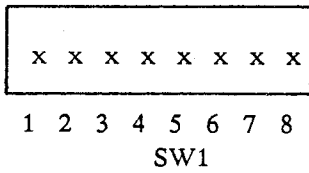
#### Brother HR-15XL



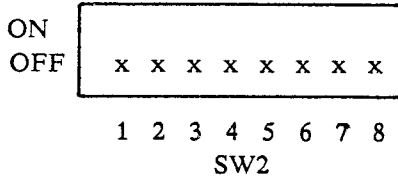
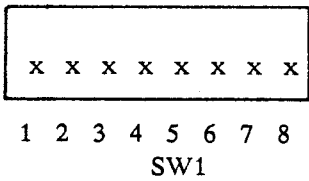
#### CBM MPS-1000



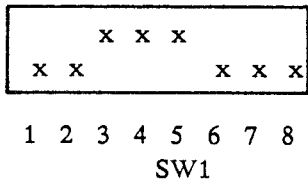
Diablo Advantage D25



Diablo 630

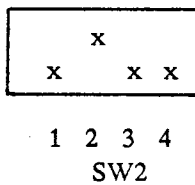
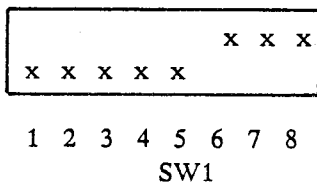


Diablo C-150

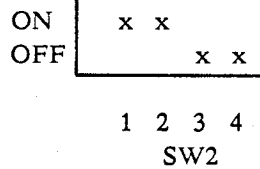
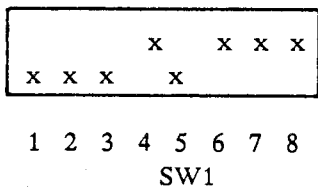


ON  
OFF

Epson LX-80



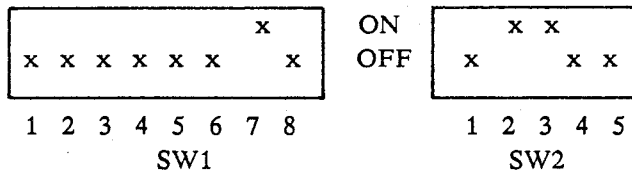
Epson JX-80



**Okimate 20**

(No switches available)

**Qume Letterpro 20P**



**HP LaserJet and LaserJet Plus**

(Switches should be set to default settings: See the Owner's Manual.)

# Appendix F

## Skeleton Device/Library Code

This appendix contains source code for a skeleton device and a skeleton library. You can use this code to create your own custom devices and libraries to add to the Amiga.

```

*
* Copyright (C) 1985, Commodore Amiga Inc. All rights reserved.
*

```

```

*
* asmsupp.i -- random low level assembly support routines
*
* Source Control
* -----
*
* $Header: asmsupp.i,v 31.1 85/10/13 23:12:33 neil Exp $
*
* $Locker: $
*

```

```

CLEAR MACRO ; quick way to clear a D register on 68000
 MOVEQ #0,\1
 ENDM

BHS MACRO \1
 BCC.\0 \1
 ENDM

BLO MACRO \1
 BCS.\0 \1
 ENDM

EVEN MACRO ; word align code stream
 DS.W 0
 ENDM

LINKSYS MACRO ; link to a library without having to see a _LVO
 LINKLIB _LVO\1,\2
 ENDM

CALLSYS MACRO ; call a library without having to see _LVO
 CALLLIB _LVO\1
 ENDM

XLIB MACRO ; define a library reference without the _LVO
 XREF _LVO\1
 ENDM

```

F-2

```

*
* Copyright (C) 1985, Commodore Amiga Inc. All rights reserved.
*

```

```

*
* mydev.i -- external declarations for skeleton device
*
* SOURCE CONTROL
* -----
*
* $Header: ramlib.i,v 31.1 85/10/13 23:12:51 neil Exp $
*
* $Locker: neil $
*

```

```

;-----
;
; device command definitions
;
;-----

DEVINIT
DEVCMD MYDEV_FOO
DEVCMD MYDEV_BAR
DEVCMD MYDEV_END ; place marker -- first illegal command #

;-----
;
; device data structures
;
;-----

; maximum number of units in this device
MD_NUMUNITS EQU 4

STRUCTURE MyDev, LIB_SIZE
 ULONG md_SysLib
 ULONG md_DosLib
 ULONG md_SegList
 UBYTE md_Flags
 UBYTE md_pad
 STRUCT md_Units, MD_NUMUNITS*4
 LABEL MyDev_Sizeof

STRUCTURE MyDevMsg, MN_SIZE
 APTR mcm_Device
 APTR mcm_Unit
 LABEL MyDevMsg_Sizeof

STRUCTURE MyDevUnit, UNIT_SIZE
 UBYTE mdu_UnitNum
 UBYTE mdu_pad
 STRUCT mdu_Msg, MyDevMsg_Sizeof
 APTR mdu_Process

```

LABEL MyDevUnit\_Sizeof

;----- state bit for unit stopped  
BITDEF MDU,STOPPED,2

; stack size and priority for the process we will create  
MYPROCSTACKSIZE EQU \$200  
MYPROCPRI EQU 0

MYDEVNAME MACRO  
DC.B 'mydev.device',0  
ENDM

\*\*\*\*\*  
\*  
\* Copyright (C) 1985, Commodore Amiga Inc. All rights reserved. \*  
\*  
\*\*\*\*\*/

\*\*\*\*\*  
\*  
\* mydev.asm -- skeleton device code  
\*  
\* Source Control  
\* -----  
\*  
\* \$Header: amain.asm,v 31.3 85/10/18 19:04:04 neil Exp \$  
\*  
\* \$Locker: neil \$  
\*  
\* \$Log: amain.asm,v \$  
\*  
\*\*\*\*\*/

SECTION section

NOLIST

include "exec/types.i"  
include "exec/nodes.i"  
include "exec/lists.i"  
include "exec/libraries.i"  
include "exec/devices.i"  
include "exec/io.i"  
include "exec/alerts.i"  
include "exec/initializers.i"  
include "exec/memory.i"  
include "exec/resident.i"  
include "exec/ables.i"  
include "exec/errors.i"  
include "libraries/dos.i"  
include "libraries/dosextens.i"

include "asmsupp.i"

include "mydev.i"

LIST

;----- These don't have to be external, but it helps some  
;----- debuggers to have them globally visible

XDEF Init  
XDEF Open  
XDEF Close  
XDEF Expunge  
XDEF Null  
XDEF myName  
XDEF BeginIO  
XDEF AbortIO

XREF \_AbsExecBase

```

XLIB OpenLibrary
XLIB CloseLibrary
XLIB Alert
XLIB FreeMem
XLIB Remove
XLIB FindTask
XLIB AllocMem
XLIB CreateProc
XLIB PutMsg
XLIB RemTask
XLIB ReplyMsg
XLIB Signal
XLIB GetMsg
XLIB Wait
XLIB WaitPort
XLIB AllocSignal
XLIB SetTaskPri

```

INT\_ABLES

```

; The first executable location. This should return an error
; in case someone tried to run you as a program (instead of
; loading you as a library).

```

```

FirstAddress:
CLEAR d0
rts

```

```

; A romtag structure. Both "exec" and "ramlib" look for
; this structure to discover magic constants about you
; (such as where to start running you from...)

```

```

; Most people will not need a priority and should leave it at zero.
; the RT_PRI field is used for configuring the roms. Use "mods" from
; wack to look at the other romtags in the system

```

```

MYPRI EQU 0

```

initDDescrip:

```

 ;STRUCTURE RT,0
DC.W RTC_MATCHWORD ; UWORD RT_MATCHWORD
DC.L initDDescrip ; APTR RT_MATCHTAG
DC.L EndCode ; APTR RT_ENDSKIP
DC.B RTE_AUTOINIT ; UBYTE RT_FLAGS
DC.B VERSION ; UBYTE RT_VERSION
DC.B NT_DEVICE ; UBYTE RT_TYPE
DC.B MYPRI ; BYTE RT_PRI
DC.L myName ; APTR RT_NAME
DC.L idString ; APTR RT_IDSTRING
DC.L Init ; APTR RT_INIT
 ; LABEL RT_SIZE

```

```

; this is the name that the device will have

```

```

subSysName:
myName: MYDEVNAME

```

```

; a major version number.
VERSION: EQU 1

```

```

; A particular revision. This should uniquely identify the bits in the
; device. I use a script that advances the revision number each time
; I recompile. That way there is never a question of which device
; that really is.

```

```

REVISION: EQU 17

```

```

; this is an identifier tag to help in supporting the device
; format is 'name version.revision (dd MON yyyy)', <cr>, <lf>, <null>
idString: dc.b 'mydev 1.0 (31 Oct 1985)', 13, 10, 0

```

```

dosName: DOSNAME

```

```

; force word alignment
ds.w 0

```

```

; The romtag specified that we were "RTE_AUTOINIT". This means
; that the RT_INIT structure member points to one of these
; tables below. If the AUTOINIT bit was not set then RT_INIT
; would point to a routine to run.

```

Init:

```

DC.L MyDev_Sizeof ; data space size
DC.L funcTable ; pointer to function initializers
DC.L dataTable ; pointer to data initializers
DC.L initRoutine ; routine to run

```

funcTable:

```

;----- standard system routines
dc.l Open
dc.l Close
dc.l Expunge
dc.l Null

```

```

;----- my device definitions
dc.l BeginIO
dc.l AbortIO

```

```

;----- function table end marker
dc.l -1

```

```

; The data table initializes static data structures.
; The format is specified in exec/InitStruct routine's
; manual pages. The INITBYTE/INITWORD/INITLONG routines
; are in the file "exec/initializers.i". The first argument
; is the offset from the device base for this byte/word/long.
; The second argument is the value to put in that cell.
; The table is null terminated

```

dataTable:

```

INITBYTE LH_TYPE, NT_DEVICE
INITLONG LN_NAME, myName
INITBYTE LIB_FLAGS, LIB_SUMUSED!LIB_CHANGED
INITWORD LIB_VERSION, VERSION
INITWORD LIB_REVISION, REVISION
INITLONG LIB_IDSTRING, idString
DC.L 0

```

```

; This routine gets called after the device has been allocated.
; The device pointer is in D0. The segment list is in a0.
; If it returns non-zero then the device will be linked into
; the device list.
initRoutine:
;----- get the device pointer into a convenient A register
move.l a5,-(sp)
move.l d0,a5

;----- save a pointer to exec
move.l a6,md_SysLib(a5)

;----- save a pointer to our loaded code
move.l a0,md_SegList(a5)

;----- open the dos library
lea dosName(pc),a1
CLEAR d0
CALLSYS OpenLibrary

move.l d0,md_DosLib(a5)
bne.s init_DosOK

;----- can't open the dos! what gives
ALERT AG_OpenLib!AO_DOSLib

init_DosOK:
;----- now build the static data that we need

;
; put your initialization here...
;

move.l a5,d0
move.l (sp)+,a5
rts

;-----
; here begins the system interface commands. When the user calls
; OpenLibrary/CloseLibrary/RemoveLibrary, this eventually gets translated
; into a call to the following routines (Open/Close/Expunge). Exec
; has already put our device pointer in a6 for us. Exec has turned
; off task switching while in these routines (via Forbid/Permit), so
; we should not take too long in them.
;-----

; Open sets the IO_ERROR field on an error. If it was successfull,
; we should set up the IO_UNIT field.

Open:
; (device:a6, iob:a1, unitnum:d0, flags:d1)
movem.l d2/a2/a3/a4,-(sp)

move.l a1,a2 ; save the iob

;----- see if the unit number is in range
moveq #MD_NUMUNITS,d2
cmp.l d2,d0
bcc.s Open_Error ; unit number out of range

;----- see if the unit is already initialized
move.l d0,d2 ; save unit number
lsl.l #2,d0
lea.l md_Units(a6,d0.l),a4
move.l (a4),d0
bne.s Open_UnitOK

;----- try and conjure up a unit
bsr InitUnit

;----- see if it initialized OK
move.l (a4),d0
beq.s Open_Error

Open_UnitOK:
move.l d0,a3 ; unit pointer in a3

move.l d0,IO_UNIT(a2)

;----- mark us as having another opener
addq.w #1,LIB_OPENCNT(a6)
addq.w #1,UNIT_OPENCNT(a3)

;----- prevent delayed expunges
bclr #LIBB_DELEXP,md_Flags(a6)

Open_End:
movem.l (sp)+,d2/a2/a3/a4
rts

Open_Error:
move.b #IOERR_OPENFAIL,IO_ERROR(a2)
bra.s Open_End

; There are two different things that might be returned from
; the Close routine. If the device is no longer open and
; there is a delayed expunge then Close should return the
; segment list (as given to Init). Otherwise close should
; return NULL.

Close:
; (device:a6, iob:a1)
movem.l a2/a3,-(sp)

move.l a1,a2

move.l IO_UNIT(a2),a3

;----- make sure the iob is not used again
moveq.l #-1,d0
move.l d0,IO_UNIT(a2)
move.l d0,IO_DEVICE(a2)

;----- see if the unit is still in use
subq.w #1,UNIT_OPENCNT(a3)

```



```

bne.s Close_Device

bsr ExpungeUnit

Close_Device:
;----- mark us as having one fewer openers
subq.w #1,LIB_OPENCNT(a6)

;----- see if there is anyone left with us open
bne.s Close_End

;----- see if we have a delayed expunge pending
btst #LIBB_DELEXP,md_Flags(a6)
beq.s Close_End

;----- do the expunge
bsr Expunge

Close_End:
movem.l (sp)+,a2/a3
rts

; There are two different things that might be returned from
; the Expunge routine. If the device is no longer open
; then Expunge should return the segment list (as given to
; Init). Otherwise Expunge should set the delayed expunge
; flag and return NULL.
;
; One other important note: because Expunge is called from
; the memory allocator, it may NEVER Wait() or otherwise
; take long time to complete.

Expunge: ; (device: a6)

movem.l d2/a5/a6,-(sp)
move.l a6,a5
move.l md_SysLib(a5),a6

;----- see if anyone has us open
tst.w LIB_OPENCNT(a5)
beq 1$

;----- it is still open. set the delayed expunge flag
bset #LIBB_DELEXP,md_Flags(a5)
CLEAR d0
bra.s Expunge_End

1$:
;----- go ahead and get rid of us. Store our seglist in d2
move.l md_SegList(a5),d2

;----- unlink from device list
move.l a5,a1
CALLSYS Remove

;
; device specific closings here...
;

```

```

;----- close the dos library
move.l md_DosLib(a5),a1
CALLSYS CloseLibrary

```

```

;----- free our memory
CLEAR d0
move.l a5,a1
move.w LIB_NEGSIZE(a5),d0

```

```

sub.w d0,a1
add.w LIB_POSSIZE(a5),d0

```

```
CALLSYS FreeMem
```

```

;----- set up our return value
move.l d2,d0

```

```

Expunge_End:
movem.l (sp)+,d2/a5/a6
rts

```

```

Null:
CLEAR d0
rts

```

```

InitUnit: ; (d2:unit number, a3:scratch, a6:devptr)
movem.l d2/d3/d4,-(sp)

```

```

;----- allocate unit memory
move.l #MyDevUnit_Sizeof,d0
move.l #MEMF_PUBLIC!MEMF_CLEAR,d1
LINKSYS AllocMem,md_SysLib(a6)

```

```

tst.l d0
beq InitUnit_End

```

```

move.l d0,a3
move.b d2,mdu_UnitNum(a3) ; initialize unit number

```

```

;----- start up the unit process. We do a trick here --
;----- we set his message port to PA_IGNORE until the
;----- new process has a change to set it up.
;----- We cannot go to sleep here: it would be very nasty
;----- if someone else tried to open the unit
;----- (exec's OpenDevice has done a Forbid() for us --
;----- we depend on this to become single threaded).

```

```

move.l #MYPROCSTACKSIZE,d4 ; stack size
move.l #myproc_seglist,d3 ; segment list
lsr.l #2,d3 ; change to bcpl pointer
moveq #MYPROCPRI,d2 ; pick out its priority
move.l #myName,d1 ; name is the device's
LINKSYS CreateProc,md_DosLib(a6)

```

```

tst.l d0
beq InitUnit_FreeUnit

```

```

;----- set up the unit structures for the new process
move.l d0,mdu_Process(a3)

```

```

move.l d0,a0
lea -pr_MsgPort(a0),a0
move.l a0,MP_SIGTASK(a3)
move.b #PA_IGNORE,MP_FLAGS(a3)

;----- send a startup message to the new process
lea mdu_Msg(a3),a1
move.l a3,mdm_Unit(a1)
move.l a6,mdm_Device(a1)
move.l d0,a0 ; message port is new process port
LINKSYS PutMsg,md_SysLib(a6)

;----- mark us as ready to go
move.l d2,d0 ; unit number
lsl.l #2,d0
move.l a3,md_Units(a6,d0.1) ; set unit table

```

```

InitUnit_End:
movem.l (sp)+,d2/d3/d4
rts

```

```

;----- got an error. free the unit structure that we allocated.
InitUnit_FreeUnit:
bsr FreeUnit
bra.s InitUnit_End

```

```

FreeUnit: ; (a3:unitptr, a6:deviceptr)
move.l a3,a1
move.l #MyDevUnit_Sizeof,d0
LINKSYS FreeMem,md_SysLib(a6)
rts

```

```

ExpungeUnit: ; (a3:unitptr, a6:deviceptr)
move.l d2,-(sp)

```

```

;----- get rid of the unit's task. We know this is safe
;----- because the unit has an open count of zero, so it
;----- is 'guaranteed' not in use.
move.l mdu_Process(a3),a1
lea -(pr_MsgPort)(a1),a1
LINKSYS RemTask,md_SysLib(a6)

```

```

;----- save the unit number
CLEAR d2
move.b mdu_UnitNum(a3),d2

```

```

;----- free the unit structure.
bsr FreeUnit

```

```

;----- clear out the unit vector in the device
lsl.l #2,d2
clr.l md_Units(a6,d2.1)

```

```

move.l (sp)+,d2
rts

```

```

;
; here begins the device specific functions
;
;-----

```

```

; cmdtable is used to look up the address of a routine that will
; implement the device command.
cmdtable:

```

|      |         |   |            |
|------|---------|---|------------|
| DC.L | Invalid | ; | \$00000001 |
| DC.L | MyReset | ; | \$00000002 |
| DC.L | Read    | ; | \$00000004 |
| DC.L | Write   | ; | \$00000008 |
| DC.L | Update  | ; | \$00000010 |
| DC.L | Clear   | ; | \$00000020 |
| DC.L | MyStop  | ; | \$00000040 |
| DC.L | Start   | ; | \$00000080 |
| DC.L | Flush   | ; | \$00000100 |
| DC.L | Foo     | ; | \$00000200 |
| DC.L | Bar     | ; | \$00000400 |

```
cmdtable_end:
```

```

; this define is used to tell which commands should not be queued
; command zero is bit zero.
; The immediate commands are Invalid, Reset, Stop, Start, Flush
IMMEDIATES EQU $000001c3

```

```

;
; BeginIO starts all incoming io. The IO is either queued up for the
; unit task or processed immediately.
;

```

```

BeginIO: ; (iob: a1, device:a6)
move.l a3,-(sp)

```

```

;----- bookkeeping
move.l IO_UNIT(a1),a3

```

```

;----- see if the io command is within range
move.w IO_COMMAND(a1),d0
cmp.w #MYDEV_END,d0
bcc.s BeginIO_NoCmd

```

```
DISABLE a0
```

```

;----- process all immediate commands no matter what
move.w #IMMEDIATES,d1
btst d0,d1
bne.s BeginIO_Immediate

```

```

;----- see if the unit is STOPPED. If so, queue the msg.
btst #MDUB_STOPPED,UNIT_FLAGS(a3)
bne.s BeginIO_QueueMsg

```

```

;----- this is not an immediate command. see if the device is
;----- busy.
bset #UNITB_ACTIVE,UNIT_FLAGS(a3)
beq.s BeginIO_Immediate

```

```

;----- we need to queue the device. mark us as needing
;----- task attention. Clear the quick flag

```

```

BeginIO_QueueMsg:
 BSET #UNITB_INTASK,UNIT_FLAGS(a3)
 bclr #IOB_QUICK,IO_FLAGS(a1)

 ENABLE a0

 move.l a3,a0
 LINKSYS PutMsg,md_SysLib(a6)
 bra.s BeginIO_End

```

```

BeginIO_Immediate:
 ENABLE a0

 bsr PerformIO

```

```

BeginIO_End:
 move.l (sp)+,a3
 rts

```

```

BeginIO_NoCmd:
 move.b #IOERR_NOCMD,IO_ERROR(a1)
 bra.s BeginIO_End

```

```

; PerformIO actually dispatches an io request. It expects a3 to already
; have the unit pointer in it. a6 has the device pointer (as always).
; a1 has the io request. Bounds checking has already been done on
; the io request.

```

```

PerformIO: ; (iob:a1, unitptr:a3, devptr:a6)
 move.l a2,-(sp)
 move.l a1,a2

 move.w IO_COMMAND(a2),d0
 lea cmdtable(pc),a0
 move.l 0(a0,d0.w),a0

 jsr (a0)

 move.l (sp)+,a2
 rts

```

```

; TermIO sends the IO request back to the user. It knows not to mark
; the device as inactive if this was an immediate request or if the
; request was started from the server task.

```

```

TermIO: ; (iob:a1, unitptr:a3, devptr:a6)
 move.w IO_COMMAND(a1),d0
 move.w #IMMEDIATES,d1
 btst d0,d1
 bne.s TermIO_Immediate

 ;----- we may need to turn the active bit off.
 btst #UNITB_INTASK,UNIT_FLAGS(a3)
 bne.s TermIO_Immediate

```

```

;----- the task does not have more work to do
bclr #UNITB_ACTIVE,UNIT_FLAGS(a3)

```

```

TermIO_Immediate:
 ;----- if the quick bit is still set then we don't need to reply
 ;----- msg -- just return to the user.
 btst #IOB_QUICK,IO_FLAGS(a1)
 bne.s TermIO_End

```

```

LINKSYS ReplyMsg,md_SysLib(a6)

```

```

TermIO_End:
 rts

```

```

AbortIO: ; (iob: a1, device:a6)

```

```

;-----
; here begins the functions that implement the device commands
; all functions are called with:
; a1 -- a pointer to the io request block
; a2 -- another pointer to the iob
; a3 -- a pointer to the unit
; a6 -- a pointer to the device
; Commands that conflict with 68000 instructions have a "My" prepended
; to them.
;-----

```

```

Invalid:
 move.b #IOERR_NOCMD,IO_ERROR(a1)
 bsr TermIO
 rts

```

```

MyReset:
 ; !!! fill me in !!!
 ; !!! fill me in !!!
 ; !!! fill me in !!!
 ; !!! fill me in !!!

```

```

; the Read command acts as an infinite source of nulls. It clears
; the user's buffer and marks that many bytes as having been read.

```

```

Read:
 move.l IO_DATA(a1),a0
 move.l IO_LENGTH(a1),d0
 move.l d0,IO_ACTUAL(a1)

 ;----- deal with a zero length read
 beq.s Read_End

```

```

;----- now copy the data
CLEAR d1

```

```

Read_Loop:

```

```

 move.b d1,(a0)+
 subq.l #1,d0
 bne.s Read_Loop

Read_End:
 bsr TermIO
 rts

;
; the Write command acts as bit bucket. It clears acknowledges all
; the bytes the user has tried to write to it.
;
Write:
 move.l IO_LENGTH(a1),IO_ACTUAL(a1)

 bsr TermIO
 rts

;
; Update and Clear are internal buffering commands. Update forces all
; io out to its final resting spot, and does not return until this is
; done. Clear invalidates all internal buffers. Since this device
; has no internal buffers, these commands do not apply.
;
Update:
Clear:
 bra Invalid

;
; the Stop command stop all future io requests from being
; processed until a Start command is received. The Stop
; command is NOT stackable: e.g. no matter how many stops
; have been issued, it only takes one Start to restart
; processing.
;
MyStop:
 bset #MDUB_STOPPED,UNIT_FLAGS(a3)

 bsr TermIO
 rts

Start:
 bsr InternalStart

 move.l a2,a1
 bsr TermIO

 rts

InternalStart:
;----- turn processing back on
bclr #MDUB_STOPPED,UNIT_FLAGS(a3)

;----- kick the task to start it moving
move.l a3,a1
CLEAR d0

 move.l MP_SIGBIT(a3),d1
 bset d1,d0
LINKSYS Signal,md_SysLib(a3)

 rts

;
; Flush pulls all io requests off the queue and sends them back.
; We must be careful not to destroy work in progress, and also
; that we do not let some io requests slip by.
;
; Some funny magic goes on with the STOPPED bit in here. Stop is
; defined as not being reentrant. We therefore save the old state
; of the bit and then restore it later. This keeps us from
; needing to DISABLE in flush. It also fails miserably if someone
; does a start in the middle of a flush.
;
Flush:
 movem.l d2/a6,-(sp)

 move.l md_SysLib(a6),a6

 bset #MDUB_STOPPED,UNIT_FLAGS(a3)
 sne d2

Flush_Loop:
 move.l a3,a0
 CALLSYS GetMsg

 tst.l d0
 beq.s Flush_End

 move.l d0,a1
 move.b #IOERR_ABORTED,IO_ERROR(a1)
 CALLSYS ReplyMsg

 bra.s Flush_Loop

Flush_End:

 move.l d2,d0
 movem.l (sp)+,d2/a6

 tst.b d0
 beq.s 1$

 bsr InternalStart

1$:
 move.l a2,a1
 bsr TermIO

 rts

;
; Foo and Bar are two device specific commands that are provided just
; to show you how to add your own commands. The currently return that
; no work was done.
;

```

```

Foo:
Bar:
 CLEAR d0
 move.l d0,IO_ACTUAL(a1)

 bsr TermIO
 rts

```

```

; here begins the process related routines
;
; A Process is provided so that queued requests may be processed at
; a later time.
;
; Register Usage
;
; a3 -- unit pointer
; a6 -- syslib pointer
; a5 -- device pointer
; a4 -- task (NOT process) pointer
; d7 -- wait mask
;

```

```

; some dos magic. A process is started at the first executable address
; after a segment list. We hand craft a segment list here. See the
; the DOS technical reference if you really need to know more about this.

```

```

 cnop 0,4 ; long word align
 DC.L 16 ; segment length -- any number will do
myproc_seglist:
 DC.L 0 ; pointer to next segment

```

```

; the next instruction after the segment list is the first executable address

```

```

Proc_Begin:

```

```

 move.l _AbsExecBase,a6

;----- wait for our first packet
SUB.L a1,a1 ; <my task> = FindTask(0)
CALLSYS FindTask
 move.l d0,a0
 move.l d0,a4 ; save task in a4
 lea pr_MsgPort(a0),a0 ; get msg port for my processes
 CALLSYS WaitPort

;----- take msg off the port
 move.l d0,a1
 move.l d0,a2 ; save the message
 CALLSYS Remove

;----- get our parameters out of it
 move.l mdm_Device(a2),a5 ; a5 is now our device
 move.l mdm_Unit(a2),a3

;----- Allocate the right signal

```

```

 moveq #-1,d0 ; -1 is any signal at all
 CALLSYS AllocSignal

```

```

 move.b d0,MP_SIGBIT(a3)
 move.b #PA_SIGNAL,MP_FLAGS(a3)

```

```

;----- change the bit number into a mask, and save in d7
 CLEAR d7
 bset d0,d7

```

```

;-----
; OK, kids, we are done with initialization. We now
; can start the main loop of the driver. It goes
; like this. Because we had the port marked PA_IGNORE
; for a while (in InitUnit) we jump to the getmsg
; code on entry.
;-----

```

```

;-----
; wait for a message
; lock the device
; get a message. if no message unlock device and loop
; dispatch the message
; loop back to get a message
;-----

```

```

 bra.s Proc_CheckStatus

```

```

;----- main loop: wait for a new message
Proc_MainLoop:
 move.l d7,d0
 CALLSYS Wait

```

```

Proc_CheckStatus:
;----- see if we are stopped
 btst #MDUB_STOPPED,UNIT_FLAGS(a3)
 bne.s Proc_MainLoop ; device is stopped

;----- lock the device
 bset #UNITB_ACTIVE,UNIT_FLAGS(a3)
 bne.s Proc_MainLoop ; device in use

```

```

;----- get the next request
Proc_NextMessage:
 move.l a3,a0
 CALLSYS GetMsg
 tst.l d0
 beq.s Proc_Unlock ; no message?

;----- do this request
 move.l d0,a1
 exg a5,a6 ; put device ptr in right place
 bsr PerformIO
 exg a5,a6 ; get syslib back in a6

 bra.s Proc_NextMessage

```

```

;----- no more messages. back ourselves out.
Proc_Unlock:
 and.b #$ff&(UNITB_ACTIVE!UNITB_INTASK),UNIT_FLAGS(a3)
 bra Proc_MainLoop

```

```

Proc_Fail:
;----- we come here on initialization failures
bsr FreeUnit
rts

```

```

; EndCode is a marker that show the end of your code.
; Make sure it does not span sections nor is before the
; rom tag in memory! It is ok to put it right after
; the rom tag -- that way you are always safe. I put
; it here because it happens to be the "right" thing
; to do, and I know that it is safe in this case.

```

```

EndCode:
 END

```

```

*
* Copyright (C) 1985, Commodore Amiga Inc. All rights reserved.
*
*****/

*
* testdev.asm -- test the mylib.asm code
*
* Source Control
* -----
*
* $Header: amain.asm,v 31.3 85/10/18 19:04:04 neil Exp $
* $Locker: neil $
* $Log: amain.asm,v $
*
*****/

```

```

INCLUDE 'exec/types.i'
INCLUDE 'exec/libraries.i'
INCLUDE 'exec/devices.i'
INCLUDE 'exec/io.i'

```

```

INCLUDE 'asmsupp.i'
INCLUDE 'mydev.i'

```

```

XDEF _main

```

```

XREF _printf
XREF _AbsExecBase
XREF _CreatePort
XREF _DeletePort
XREF _CreateStdIO
XREF _DeleteStdIO

```

```

XLIB OpenDevice
XLIB CloseDevice

```

```

_main:
move.l _AbsExecBase,a6

;----- make a reply port
pea 0
pea myName
jsr _CreatePort
addq.l #8,sp

move.l d0,Port
beq.s main_end

;----- get an io request
move.l d0,-(sp)
jsr _CreateStdIO
addq.l #4,sp

move.l d0,Iob
beq main_DeletePort

```

```

move.l d0,a1
move.l #myName,LN_NAME(a1)

;----- open the test device: this will bring it in from disk
lea myDevName(pc),a0
CLEAR d0
CLEAR d1
CALLSYS OpenDevice

tst.l d0
beq.s 1$

;----- couldn't find the library
pea 0
move.l d0,a0
move.b IO_ERROR(a0),3(sp)
pea myDevName(pc)
pea nodevmsg(pc)
jsr _printf
addq.l #8,sp

bra main_DeleteIob

1$:

;----- close the device
move.l Iob,a1
CALLSYS CloseDevice

main_DeleteIob:
move.l Iob,-(sp)
jsr _DeleteStdIO
addq.l #4,sp

main_DeletePort
move.l Port,-(sp)
jsr _DeletePort
addq.l #4,sp

main_end:
rts

myDevName: MYDEVNAME
myName: dc.b 'testdev',0
nodevmsg: dc.b 'can not open device "%s": error %ld',10,0
testmsg: dc.b 'function MYFUNC%ld returned %ld',10,0

Port: dc.l 0
Iob: dc.l 0

END

```

```

*
* Copyright (C) 1985, Commodore Amiga Inc. All rights reserved.
*

```

```

*
* mylib.i -- external declarations for skeleton library
*
* SOURCE CONTROL
* -----
* $Header: ramlib.i,v 31.1 85/10/13 23:12:51 neil Exp $
*
* $Locker: neil $
*

```

```

;-----
;
; library function definitions
;
;-----

```

```

LIBINIT
LIBDEF MLFUNC0
LIBDEF MLFUNC1

```

```

;-----
;
; library data structures
;
;-----

```

```

STRUCTURE MyLib,LIB_SIZE
ULONG ml_SysLib
ULONG ml_DosLib
ULONG ml_SegList
UBYTE ml_Flags
UBYTE ml_pad
LABEL MyLib_Sizeof

```

```

MYLIBNAME
MACRO
DC.B 'mylib.library',0
ENDM

```

```

*
* Copyright (C) 1985, Commodore Amiga Inc. All rights reserved.
*
*****/

*
* mylib.asm -- skeleton library code
*
* Source Control
* -----
*
* $Header: amain.asm,v 31.3 85/10/18 19:04:04 neil Exp $
* $Locker: neil $
* $Log: amain.asm,v $
*
*****/
SECTION section

NOLIST
include "exec/types.i"
include "exec/nodes.i"
include "exec/lists.i"
include "exec/libraries.i"
include "exec/alerts.i"
include "exec/initializers.i"
include "exec/resident.i"
include "libraries/dos.i"

include "asmsupp.i"

include "mylib.i"

LIST

;----- These don't have to be external, but it helps some
;----- debuggers to have them globally visible
XDEF Init
XDEF Open
XDEF Close
XDEF Expunge
XDEF Null
XDEF myName
XDEF MyFunc0
XDEF MyFunc1

XREF _AbsExecBase

XLIB OpenLibrary
XLIB CloseLibrary
XLIB Alert
XLIB FreeMem
XLIB Remove

; The first executable location. This should return an error
; in case someone tried to run you as a program (instead of
; loading you as a library).

```

```

Start:
 CLEAR d0
 rts

; A romtag structure. Both "exec" and "ramlib" look for
; this structure to discover magic constants about you
; (such as where to start running you from...).

; Most people will not need a priority and should leave it at zero.
; the RT_PRI field is used for configuring the roms. Use "mods" from
; wack to look at the other romtags in the system
MYPRI EQU 0

initDDescrip:
 DC.W RTC_MATCHWORD ; UWORD RT_MATCHWORD
 DC.L initDDescrip ; APTR RT_MATCHTAG
 DC.L EndCode ; APTR RT_ENDSKIP
 DC.B RTE_AUTOINIT ; UBYTE RT_FLAGS
 DC.B VERSION ; UBYTE RT_VERSION
 DC.B NT_LIBRARY ; UBYTE RT_TYPE
 DC.B MYPRI ; BYTE RT_PRI
 DC.L myName ; APTR RT_NAME
 DC.L idString ; APTR RT_IDSTRING
 DC.L Init ; APTR RT_INIT
 ; LABEL RT_SIZE

myName: ; this is the name that the library will have
 MYLIBNAME

 ; a major version number.
VERSION: EQU 1

 ; A particular revision. This should uniquely identify the bits in the
 ; library. I use a script that advances the revision number each time
 ; I recompile. That way there is never a question of which library
 ; that really is.
REVISION: EQU 17

 ; this is an identifier tag to help in supporting the library
 ; format is 'name version.revision (dd MON yyyy)', <cr>, <lf>, <null>
idString: DC.B 'mylib 1.0 (31 Oct 1985)', 13, 10, 0

dosName: DOSNAME

 ; force word allignment
ds.w 0

 ; The romtag specified that we were "RTE_AUTOINIT". This means
 ; that the RT_INIT structure member points to one of these
 ; tables below. If the AUTOINIT bit was not set then RT_INIT
 ; would point to a routine to run.

Init:
 DC.L MyLib_Sizeof ; data space size
 DC.L funcTable ; pointer to function initializers

```



```

DC.L dataTable ; pointer to data initializers
DC.L initRoutine ; routine to run

```

funcTable:

```

;----- standard system routines
dc.l Open
dc.l Close
dc.l Expunge
dc.l Null

;----- my libraries definitions
dc.l MyFunc0
dc.l MyFunc1

;----- function table end marker
dc.l -1

```

```

; The data table initializes static data structures.
; The format is specified in exec/InitStruct routine's
; manual pages. The INITBYTE/INITWORD/INITLONG routines
; are in the file "exec/initializers.1". The first argument
; is the offset from the library base for this byte/word/long.
; The second argument is the value to put in that cell.
; The table is null terminated

```

dataTable:

```

INITBYTE LH_TYPE,NT_LIBRARY
INITLONG LN_NAME,myName
INITBYTE LIB_FLAGS,LIBF_SUMUSED!LIBF_CHANGED
INITWORD LIB_VERSION,VERSION
INITWORD LIB_REVISION,REVISION
INITLONG LIB_IDSTRING,idString
DC.L 0

```

```

; This routine gets called after the library has been allocated.
; The library pointer is in D0. The segment list is in A0.
; If it returns non-zero then the library will be linked into
; the library list.

```

initRoutine:

```

;----- get the library pointer into a convenient A register
move.l a5,-(sp)
move.l d0,a5

;----- save a pointer to exec
move.l a6,m1_SysLib(a5)

;----- save a pointer to our loaded code
move.l a0,m1_SegList(a5)

;----- open the dos library
lea dosName(pc),a1
CLEAR d0
CALLSYS OpenLibrary

move.l d0,m1_DosLib(a5)
bne.s 1$

```

```

;----- can't open the dos! what gives
ALERT AG_OpenLib!AO_DOSLib

```

1\$:

```

;----- now build the static data that we need

```

```

;
; put your initialization here...
;

```

```

move.l a5,d0
move.l (sp)+,a5
rts

```

```

;-----
;
; here begins the system interface commands. When the user calls
; OpenLibrary/CloseLibrary/RemoveLibrary, this eventually gets translated
; into a call to the following routines (Open/Close/Expunge). Exec
; has already put our library pointer in A6 for us. Exec has turned
; off task switching while in these routines (via Forbid/Permit), so
; we should not take too long in them.
;-----

```

```

; Open returns the library pointer in d0 if the open
; was successful. If the open failed then null is returned.
; It might fail if we allocated memory on each open, or
; if only open application could have the library open
; at a time...

```

Open: ; ( libptr:a6, version:d0 )

```

;----- mark us as having another opener
addq.w #1,LIB_OPENCNT(a6)

```

```

;----- prevent delayed expunges
bclr #LIBB_DELEXP,m1_Flags(a6)

```

```

move.l a6,d0
rts

```

```

; There are two different things that might be returned from
; the Close routine. If the library is no longer open and
; there is a delayed expunge then Close should return the
; segment list (as given to Init). Otherwise close should
; return NULL.

```

Close: ; ( libptr:a6 )

```

;----- set the return value
CLEAR d0

```

```

;----- mark us as having one fewer openers
subq.w #1,LIB_OPENCNT(a6)

```

```

;----- see if there is anyone left with us open
bne.s 1$

```

```

;----- see if we have a delayed expunge pending
btst #LIBB_DELEXP,ml_Flags(a6)
beq.s 1$

;----- do the expunge
bsr Expunge

1$:
rts

; There are two different things that might be returned from
; the Expunge routine. If the library is no longer open
; then Expunge should return the segment list (as given to
; Init). Otherwise Expunge should set the delayed expunge
; flag and return NULL.
;
; One other important note: because Expunge is called from
; the memory allocator, it may NEVER Wait() or otherwise
; take long time to complete.

Expunge: ; (libptr: a6)

movem.l d2/a5/a6,-(sp)
move.l a6,a5
move.l ml_SysLib(a5),a6

;----- see if anyone has us open
tst.w LIB_OPENCNT(a5)
beq 1$

;----- it is still open. set the delayed expunge flag
bset #LIBB_DELEXP,ml_Flags(a5)
CLEAR d0
bra.s Expunge_End

1$:
;----- go ahead and get rid of us. Store our seglist in d2
move.l ml_SegList(a5),d2

;----- unlink from library list
move.l a5,a1
CALLSYS Remove

;
; device specific closings here...
;

;----- close the dos library
move.l ml_DosLib(a5),a1
CALLSYS CloseLibrary

;----- free our memory
CLEAR d0
move.l a5,a1
move.w LIB_NEGSIZE(a5),d0

sub.l d0,a1
add.w LIB_POSSIZE(a5),d0

CALLSYS FreeMem

```

```

;----- set up our return value
move.l d2,d0

Expunge_End:
movem.l (sp)+,d2/a5/a6
rts

Null:
CLEAR d0
rts

;-----
; here begins the library specific commands
;-----

MyFunc0:
CLEAR d0
rts

MyFunc1:
moveq #1,d0
rts

; EndCode is a marker that show the end of your code.
; Make sure it does not span sections nor is before the
; rom tag in memory! It is ok to put it right after
; the rom tag -- that way you are always safe. I put
; it here because it happens to be the "right" thing
; to do, and I know that it is safe in this case.

EndCode:

END

```

```

INCLUDE 'exec/types.i'
INCLUDE 'exec/libraries.i'

INCLUDE 'asmsupp.i'
INCLUDE 'mylib.i'

```

```

*
* Copyright (C) 1985, Commodore Amiga Inc. All rights reserved. *
*
*****/

```

```

*
* testlib.asm -- test the mylib.asm code
*
* Source Control
* -----
*
* $Header: amain.asm,v 31.3 85/10/18 19:04:04 neil Exp $
*
* $Locker: neil $
*
* $Log: amain.asm,v $
*
*****/

```

```

XDEF _main

XREF _printf
XREF _AbsExecBase

XLIB OpenLibrary
XLIB CloseLibrary

```

```

_main:
move.l _AbsExecBase,a6

;----- open the test library: this will bring it in from disk
lea myName(pc),a1
CLEAR d0
CALLSYS OpenLibrary

tst.l d0
bne.s l$

;----- couldn't find the library
pea myName(pc)
pea nolibmsg(pc)
jsr _printf
addq.l #8,sp

bra main_end

```

```

l$:
move.l d0,a2

;----- call the first test function
LINKLIB MLFUNC0,a2
move.l d0,-(sp)
pea 0
pea testmsg(pc)
jsr _printf
lea 12(sp),sp

;----- call the second test function
LINKLIB MLFUNC1,a2
move.l d0,-(sp)
pea 1
pea testmsg(pc)
jsr _printf
lea 12(sp),sp

;----- close the library
move.l a2,a1
CALLSYS CloseLibrary

main_end:
rts

myName: MYLIBNAME
nolibmsg: dc.b 'can not open library "%s",10,0
testmsg: dc.b 'function MYFUNC%d returned %d',10,0

END

```

# Index

- AbortIO()**, 276
- AddAnimOb()**, 169
- AddBob()**, 147
- AddTime()**, 251
- AddVSprite()**, 125
- After** pointer
  - changing Bob priority, 149
  - in animation precedence, 172
  - in Bob priority, 141-142
  - in linking AnimComps, 175
- AllocMem()**, 143
- AllocRaster()**
  - allocating memory, 29
  - in saving background, 143
- Alt key, 305, 323
- Amiga keys, 292
- AndRectRegion()**, 96
- Animate()**, 163, 174, 177
- animation
  - acceleration, 168
  - AnimCRoutine, 173
  - AnimORoutine, 173
  - motion control, 168-169
  - sequenced drawing, 166, 169
  - types, 104, 105
  - velocity, 168
- AnimComp**
  - BobComp**, 145
  - BOBISCOMP flag, 145
  - definition, 164
  - Flags** variable, 173
  - position, 165
  - TimeSet** variable, 173
- AnimCRoutine
  - in creating animation, 176
  - with **Animate()**, 177
- AnimOb**
  - definition, 164
  - position, 165
- AnimORoutine
  - in creating animation, 176
  - with **Animate()**, 177
- AnX** variable
  - in ring processing, 175
  - in velocity and acceleration, 168
  - moving registration point, 168
  - specifying registration point, 165
- AnY** variable
  - in ring processing, 175
  - in velocity and acceleration, 168
  - moving registration point, 168
  - specifying registration point, 165
- AOIPen** variable
  - in filling, 37
  - in **RastPort**, 36
- A-Pen, *see* **FgPen**
- area buffer, 41
- area pattern, 38
- AreaDraw()**
  - adding a vertex, 45
  - in area fill, 41
- AreaEnd()**
  - drawing and filling shapes, 45
  - in area fill, 41
- AreaInfo** pointer, 41
- AreaMove()**
  - beginning a polygon, 45
  - in area fill, 41
- AskKeyMap()**, 298
- AskSoftStyle()**, 199
- audio channels
  - allocation, 225, 228
  - allocation key, 226, 230
  - changing the precedence, 231
  - freeing, 230-231
- audio device
  - AbortIO()**, 228
  - allocation/arbitration commands, 228
  - BeginIO()**, 228
  - CloseDevice()**, 228
  - double-buffering, 233
  - hardware control commands, 232
  - IORequest block, 224
  - OpenDevice()**, 227
  - playing the sound, 232
  - precedence of users, 225
  - scope of commands, 224
  - starting the sound, 235
  - stopping the sound, 233-234
  - use of **BeginIO()** function, 226

**AvailFonts()**, 201  
background pen, 36  
background playfield, 31  
BDRAWN flag, 146  
beam synchronization, 58  
**Before** pointer  
  changing Bob priority, 149  
  in animation precedence, 172  
  in Bob priority, 141-142  
  in linking AnimComps, 175  
**BeginUpdate()**, 95  
**BehindLayer()**, 83  
**BgPen**, 197  
**BitMap**  
  address, 23  
  in double-buffering, 33  
  in superbitmap layers, 86  
  software clipping, 46  
  with write mask, 42  
**BitMap** structure  
  in dual-playfield display, 32  
  preparing, 23  
bit-planes  
  extracting a rectangle from, 54  
  in dual-playfield display, 30  
blitter  
  in **Bob** animation, 109  
  in copying data, 57  
  in disk driver, 262  
  VBEAM counter, 60  
**BltBitMap()**, 55  
**BltClear()**, 50  
**bltnode** structure  
  creating, 59  
  linking blitter requests, 57  
**BltPattern()**, 52  
**BltTemplate()**, 53  
BNDRYOFF macro, 46  
**BobComp** pointer, 145  
BOBISCOMP flag, 145  
BOBNIX flag, 146  
BOBSAWAY flag, 146  
**Bobs**  
  adding new features, 161  
  as a paintbrush, 145  
  as part of AnimComp, 145  
  **Before**, **After** pointers, 172  
  bit-planes, 138, 140  
  changing, 148  
  clipping, 145  
  colors, 136, 138, 140, 152  
  defining, 135  
  definition, 109  
  displaying, 148  
  double-buffering, 146, 149  
  drawing order, 141  
  list, 142  
  priorities, 141  
  removing, 146  
  saving the background, 143  
  shadow mask, 139, 144  
  shape, 137  
  size, 136  
  sorting the list, 148  
  structure, 135  
  transparency, 144  
  troubleshooting, 153  
BORDERHIT flag, 159  
**BorderLine** pointer, 157  
BOTTOMHIT flag, 154  
**bottommost** variable  
  in Bobs clipping region, 145  
  in Bob/VSprite collision, 161  
B-Pen, *see* **BgPen**  
**BufPath** variable, 150  
**BufY**, **BufX** variables, 150  
**BufBuffer** variable, 150  
BWAITING flag, 145  
**bytecnt** variable, 53  
**bytecount** pointer, 50  
Caps Lock key, 291, 304  
**ChangeSprite()**, 112  
**CheckIO()**, 276, 386, 404  
**cleanup** variable, 60  
**ClearRegion()**, 96  
**ClipBlit()**, 55  
clipping  
  in area fill, 46  
  in line drawing, 44  
  text, 193  
clipping rectangles  
  in Bob/VSprite collision, 161  
  in layer operations, 84  
  in layers, 78, 94  
  modifying regions, 96  
clipping region  
  in Bobs, 145  
  in boundary collisions, 159  
  in VSprites with GELGONE, 125  
**ClipRect** structure, 94  
**CloseDevice()**, 307  
**Close()**, 416  
CMD\_CLEAR command, 268  
CMD\_UPDATE command, 268  
CMD\_WRITE command, 268  
**CmpTime()**, 251  
collisions  
  between GEL objects, 153

- boundary, 159
- boundary hits, 154
- collision mask, 156
- detection in hardware, 153
- fast detection, 157
- GEL-to-GEL, 154
- in animation, 153
- multiple, 155
- sensitive areas, 157
- user routines, 159
- CollMask** variable
  - in Bobs, 139
  - with collision mask, 156
- color
  - affect of display mode on, 6
  - Bobs**, 136, 152
  - ColorMap** structure, 24
  - flickering, 134
  - in dual playfield mode, 15
  - in flood fill, 47
  - in hold-and-modify mode, 34
  - interaction between VSprites and Bobs, 152
  - mode in flood fill, 47
  - of individual pixel, 43
  - playfield and VSprites, 134
  - relationship to bit-planes, 8
  - relationship to depth of **BitMap**, 13
  - simple sprites, 111
  - single-color raster, 50
  - sprites, 16
  - transparency, 123
  - VSprite**, 121, 133
- ColorMap** structure, 24
- CommandTable**, 436
- compFlags** variable, 168
- COMPLEMENT, 197
- complement mode, 38
- ConMayGetChar()**, 286
- ConPutChar()**, 281
- console
  - alternate key maps, 303
  - capsable keys, 304
  - character output, 276
  - closing, 307
  - control sequence introducer, 290
  - control sequences, 281
  - high key map, 299, 306
  - input event qualifiers, 291
  - input stream, 287
  - keyboard input, 277
  - keymapping, 292, 297
  - keymapping qualifiers, 300-301
  - keytypes, 302
  - low key map, 299, 305
  - mouse button events, 297
  - raw events, 289
  - raw input types, 289
  - reads, 286
  - repeatable keys, 305
  - string output keys, 303
  - window bounds, 289
- ConWrite()**, 281
- cookie cut, 57
- Copper
  - changing colors, 24
  - display instructions, 25
  - in drawing VSprites, 122
  - in interlaced displays, 33
  - long-frame list, 33
  - MakeVPort()**, 29
  - MrgCop()**, 25, 33
  - short-frame list, 33
  - user Copper lists, 61
- copying
  - data, 57
  - rectangles, 55
- count** variable, 44
- cp\_x** variable
  - in drawing, 42
  - in text, 192
- cp\_y** variable
  - in drawing, 42
  - in text, 192
- crashing
  - with drawing routines, 44
  - with fill routines, 46
- CreateBehindLayer()**, 81-82
- CreateExtIO()**, 264, 384, 402, 417
- CreatePort()**, 265, 384, 402, 417
- CreateStdIO()**, 264, 279
- CreateUpFrontLayer()**, 81-82
- Ctrl key, 305
- DamageList** structure
  - in layers, 94
  - in regions, 95
- DBuffer** pointer, 149
- DBufPacket** structure, 150
- deallocation
  - Copper list, 30
  - memory, 30, 41
- DeleteExtIO()**, 384, 402
- DeleteLayer()**, 82
- DeletePort()**, 384, 402
- DeleteStdIO()**, 307
- depth, 13
- Depth** variable, 136, 138
- destRastPort** variable, 55
- destX** variable, 55

- destY** variable, 55
- DHeight** variable
  - in **ViewPort**, 20
  - in **ViewPort** display memory, 19
- diskfont library, 200
- diskfont.h*, 201-202
- DisownBlitter()**, 57
- display fields, 5
- display modes, 15
- display width
  - affect of overscan on, 4
  - effect of resolution on, 17
- DisposeRegion()**, 96
- DMA
  - displaying the **View**, 25
  - playfield, 13
- DoCollision()**
  - purpose, 153
  - with collision masks, 160
- DoIO()**, 276
- DoSpecial()**, 435-437
- dotted lines, 38
- double-buffering
  - allocations for, 32
  - Copper in, 33
  - Copper lists, 128
  - with **Bobs**, 149
- DrawerData** structure, 482
- Draw()**
  - in line drawing, 43
  - multiple line drawing, 44
- DrawGList()**
  - and **BDRAWN** flag, 146
  - and **BOBNIX** flag, 146
  - and **BOBSAWAY** flag, 146
  - and **BWAITING** flag, 145
  - animation, 163
  - changing **Bobs**, 149
  - displaying **Bobs**, 148
  - linking **AnimComps**, 175
  - moving registration point, 168
  - preparing the **GELS** list, 127
  - removing **Bobs**, 147
  - with **DoCollision()**, 177
- drawing
  - changing part of drawing area, 52
  - clearing memory, 50
  - colors, 37
  - complement mode, 38
  - lines, 43
  - memory for, 35
  - modes, 37-38
  - moving source to destination, 53
  - pens, 36-37
  - pixels, 43
  - shapes, 47
  - turning off outline, 46
- drawing pens
  - color, 37
  - current position, 42
- DrawMode** variable
  - in area drawing and filling, 45
  - in flood fill, 48
  - in stencil drawing, 52
  - with **BltTemplate**, 55
  - in text, 197
- dual playfields
  - bit-planes, 31
  - color map, 24
  - colors, 15
  - priority, 31
- DUALPF** flag
  - in dual playfield display, 31
  - in **ViewPort**, 15
- DumpRPort()**, 415, 423
- DWidth** variable
  - in **ViewPort**, 12-13, 20
  - in **ViewPort** display memory, 19
- DxOffset** variable
  - effect on display window, 20
  - in **ViewPort** display memory, 19
- DyOffset** variable
  - effect on display window, 20
  - in **ViewPort** display memory, 19
- EndUpdate()**, 95
- EQUAL** status code, 59
- ETD\_CLEAR** command, 268
- ETD\_MOTOR** command, 268
- ETD\_READ** command, 267
- ETD\_UPDATE** command, 268
- ETD\_WRITE** command, 268
- EXTRA\_HALFBRITE** flag, 15-16
- fast floating-point library, 453
- FattenLayerInfo()**, 80
- FgPen** variable
  - in area drawing and filling, 45
  - in complement mode, 38
  - in flood fill, 47-48
  - in **JAM1** mode, 37
  - in line drawing, 44
  - in **RastPort**, 36
  - in rectangle fill, 48
  - in text, 197
  - with **BltTemplate**, 55
- FindToolType()**, 487
- Flags** variable
  - in **AnimComps**, 173
  - in layers, 85

- in VSprites, 124
- with BNDRYOFF macro, 46
- flicker, 58, 60
- Flood()**, 47
- floppy disk, 262
- FontContents** structure, 201
- FontContentsHeader** structure, 201
- fonts, 192
- Forbid()**, 487
- foreground pen, 36
- FOREVER loop, 29
- FreeColorMap()**, 29
- FreeCprList()**, 29
- FreeDiskObject()**, 481
- FreeRaster()**, 29
- FreeSprite()**, 120
- FreeVPortCopLists()**, 29
- Gadget** structure, 483
- gameport connectors, 322
- gameport device
  - connectors, 345
  - system functions, 346
  - triggering events, 348
  - units, 322, 345
- gameport.h*, 348
- GameTrigger** structure, 348
- GELGONE flag
  - in Bobs, 145
  - with VSprites, 125
- GELS
  - initializing, 106
  - list, 106
  - types, 107
- GelsInfo** pointer, 42
- GelsInfo** structure, 131
- GetColorMap()**, 29
- GetDiskObject()**, 481
- GetMsg()**, 386, 404
- GetSprite()**, 111
- GfxBase** variable, 22
- GPD\_GETCTYPE command, 347
- GPD\_SETCTYPE command, 346
- GPD\_SETTRIGGER command, 348
- graphics library, 22
- HAM flag, 15, 34
- hardware sprites
  - allocation, 111
  - in animation, 25
  - reserving, 131
- Height** variable
  - in Bobs, 136, 138
  - in **ViewPort**, 12
  - in VSprites, 121
- HIRES flag, 15

- HitMask** variable, 159
- hold-and-modify mode, 34
- icon library, 480
- IDCMP, 327
- ImageData** pointer
  - changing Bobs, 148
  - changing VSprites, 129
  - in Bobs, 137
  - in VSprites, 122-123
- Image** structure, 484
- ImageShadow** variable
  - in Bobs, 139
  - with OVERLAY flag, 144
- IND\_ADDHANDLER command, 324
- IND\_REMHANDLER command, 326
- IND\_SETPERIOD command, 327
- IND\_SETTHRESH command, 327
- IND\_WRITEEVENT command, 326
- info file, 481
- InitArea()**, 41
- InitGels()**, 106
- InitLayers()**, 80
- InitMasks()**
  - changing Bob image shadow, 148
  - defining collision mask, 156
  - with **Borderline**, 158
- InitRastPort()**, 196
- input device
  - adding a handler, 324
  - and console device, 323
  - commands, 323
  - designing an input handler, 324
  - generating input events, 326
  - IOStdReq** block, 324
  - key repeat events, 327
  - memory deallocation, 325
  - opening, 322
  - removing a handler, 326
  - setting key repeat interval, 327
  - setting key repeat timing, 327
- input event chain, 325
- input event structure, 323
- input events
  - generators of, 326
  - Intuition handling of, 324
  - mouse button, 328
- inputevent.h*, 323
- input\_request\_block, 326
- Intuition
  - as input device handler, 324
  - mouse input, 322
  - with sprites, 110
- INVERSEVID mode
  - in drawing, 38



- in text, 198
- IODRPreq** structure, 416
- IOExtPar** structure, 403
- IOExtSer** structure, 385, 396
- IOExtTD** structure, 265
- IOPrtCmdReq** structure, 416
- IOStdReq** structure, 279
- io\_TermArray**, 388, 405
- JAM1 mode
  - in drawing, 37
  - in text, 197
  - with INVERSEVID, 38
- JAM2 mode
  - in drawing, 37
  - in text, 197
- joystick controller, 347
- KBD\_ADDRESETHANDLER** command, 339
- KBD\_READEVENT** command, 341
- KBD\_READMATRIX** command, 340
- KBD\_REMRESETHANDLER** command, 340
- KBD\_RESETHANDLERDONE** command, 340
- keyboard device
  - keyboard events, 337
  - system functions, 338
- keyboard layout, 292
- KeyMap** structure, 298
- keymap.h*, 301
- keymap.i*, 301
- LACE flag
  - in **View** and **ViewPort**, 18
  - in **ViewPort**, 15
- layer refresh
  - simple refresh, 85
  - smart refresh, 86
  - superbitmap, 86
- LAYERBACKDROP flag, 86
- Layer\_Info** structure, 80, 88
- layers
  - accessing, 81
  - backdrop, 86
  - blocking output, 81
  - clipping rectangle list, 94
  - creating, 81-82, 88
  - creating the workspace, 88
  - deleting, 82
  - moving, 82
  - order, 83
  - redrawing, 95
  - scrolling, 83
  - simple refresh, 94
  - sizing, 82
  - sub-layer operations, 84
  - updating, 95
- layers library
  - contents, 77
  - opening, 87
- LAYERSIMPLE flag, 85
- LAYERSMART flag, 85
- LAYERSUPER flag, 85
- LEFTHIT flag, 154
- leftmost** variable
  - in Bobs clipping region, 145
  - in Bob/VSprite collision, 161
- line drawing, 43
- line pattern, 38
- LinePtrn** variable, 45
- lines
  - multiple, 44
  - patterned, 44
- LoadRGB4()**, 24
- LoadView()**
  - effect of freeing memory, 30
  - in display **ViewPorts**, 25
- Locklayer()**, 81
- LockLayerInfo()**, 81
- LockLayers()**, 81
- LOFCprList** variable, 33
- logic equations, 56
- long-frame Copper list, 33
- MakeView()**
  - with simple sprites, 110
- MakeVPort()**
  - allocating memory, 29
  - in double-buffering, 33
  - in dual playfield display, 31
- Mask** variable, 42, 55
- MatchToolValue()**, 488
- math library, 453
- mathffp.library, 455
- mathieedoubbas\_lib.lib*, 474
- mathieedoubbas.library, 472
- mathlink.lib*, 457, 464
- mathtrans.library, 461
- maxx** variable, 52
- maxy** variable, 53
- MeMask** variable, 159
- memblock** pointer, 50
- memory
  - allocation for **BitMap**, 23
  - clearing, 50
  - deallocation of, 41
  - for area fill, 41
  - freeing, 29
- MICROHZ timer unit, 248
- minterm** variable, 56
- Modes** variable
  - in **View** structure, 20
  - in **ViewPort**, 14-15

- modulo, 54
- mouse button, 323
- mouse button events, 297, 322
- mouse controller, 347
- mouse movement events, 322
- mouth** structure, 365
- Move()**, 43, 192
- MoveLayer()**, 82
- MoveSprite()**, 113
- MrgCop()**
  - in graphics display, 25
  - installing VSprites, 128
  - merging Copper lists, 30
- myInfo** structure, 162
- narrator device
  - Arpabet, 373
  - consonants, 374
  - content words, 376
  - contractions, 374
  - controlling speech characteristics, 364
  - function words, 376
  - improving intelligibility, 378
  - mouth shape, 365
  - opening, 363
  - opening the device, 366
  - output buffer, 362
  - phonemes, 373
  - phonetic spelling, 373
  - punctuating phonetic strings, 372
  - punctuation, 377
  - reading and writing, 366
  - recommended stress values, 376
  - special symbols, 375
  - speech synthesis system, 379
  - stress and intonation, 375
  - stress marks, 375
  - translator library, 362
  - vowels, 373
- narrator.h*, 365
- narrator.i*, 365
- narrator\_rb** structure, 364
- NewLayerInfo()**, 80
- NewRegion()**, 95
- NewWindow** structure, 482
- NextComp** pointer
  - in linking AnimComps, 175
  - in sequenced drawing, 171
- Next** variable, 22
- NextSeq** pointer
  - in linking AnimComps, 175
  - in sequenced drawing, 170
- NOTEQUAL status code, 59
- ON\_DISPLAY macro, 127
- ON\_SPRITE macro, 127
- O-Pen, *see* AOIPen
- OpenConsole()**, 278
- OpenDiskFont()**, 195, 205
- OpenFont()**, 195, 205
- Open()**, 416
- OpenScreen()**, 110
- OrRectRegion()**, 96
- outline mode, 47
- outline pen, 36
- OVERLAY flag, 144
- OwnBlitter()**, 57
- PAR:, 414
- parallel device
  - closing, 408
  - EOF mode, 405-406
  - errors, 407
  - flags, 406
  - IOExtPar** block, 403
  - io\_TermArray**, 405-406
  - loading from disk, 402
  - opening, 402
  - opening timer device, 403
  - PDCMD\_SETPARAMS**, 403
  - reading, 403, 404
  - setting parameters, 403, 406-407
  - shared access, 406
  - terminating the read, 405
  - termination characters, 406
  - writing, 405
- PDCMD\_SETPARAMS**, 403
- PED** structure, 428
- PFBA flag
  - in dual playfield mode, 17
  - in **ViewPort**, 15
- pixel width, 17
- PlaneOnOff** variable
  - changing Bob color, 149
  - in Bobs, 140
- PlanePick** variable
  - changing Bob color, 149
  - in Bobs, 138, 140-141
- PLANEPTR, 23
- PolyDraw()**, 44
- polygons, 45
- power\_of\_two** variable, 38
- PRD\_DUMPRPORT**, 423
- PRD\_PRTCOMMAND**, 422
- Preferences, 415, 428
- PrevComp** pointer
  - in linking AnimComps, 175
  - in sequenced drawing, 171
- PrevSeq** pointer
  - in linking AnimComps, 175
  - in sequenced drawing, 170

**PrintCommand()**, 422  
 printer device  
   alphanumeric drivers, 435  
   buffer deallocation, 434  
   buffer space, 432  
   closing DOS printer device, 416  
   command buffer, 432  
   command functions, 419  
   **CommandTable**, 435  
   creating an I/O request, 417  
   creating drivers, 428  
   data structures, 416  
   density, 434  
   direct use, 415  
   DOS parallel device, 414  
   DOS printer device, 414  
   DOS serial device, 414  
   double buffering, 433  
   dumping a **RastPort**, 423  
   dumping buffer, 433  
   Exec printer I/O, 416  
   graphics printer drivers, 431  
   opening, 418  
   opening DOS printer device, 415  
   output forms, 414  
   output methods, 413  
   PAR:, 414  
   Preferences, 415, 428, 430, 437  
   printer types, 431  
   processes and tasks, 416  
   PRT:, 414, 419  
   reset command, 433  
   SER:, 414  
   timeout, 430  
   transmitting commands, 422  
   writing, 418  
**PrinterData** structure, 430  
**PrinterExtendedData** structure, 428  
**printerIO** structure, 416  
 PRT:, 414  
**PutDiskObject()**, 481  
**PWait()**, 433  
**QBlit()**  
   linking **bltnodes**, 58  
   waiting for the blitter, 57  
**QBSBlit()**  
   avoiding flicker, 58  
   linking **bltnodes**, 58  
   waiting for the blitter, 57  
**QueueRead()**, 286  
**RasInfo** structure, 20  
**RASSIZE** macro, 21  
 raster  
   depth, 13  
   dimensions, 19  
   in dual-playfield mode, 15  
   memory allocation, 21  
   one color, 50  
   **RasInfo** structure, 20  
   scrolling, 51  
**RastPort**  
   in area fill, 41  
   in layers, 84  
   pointer to, 42  
**RastPort** structure, 192  
**rastport** variable, 52  
*rastport.h*, 36  
*rastport.i*, 36  
**RawWrite()**, 415  
**ReadPixel()**, 43  
 rectangle fill, 48  
 rectangle scrolling, 51  
**RectFill()**, 48  
 regions  
   changing, 96  
   clearing, 96  
   creating, 95  
   removing, 96  
 registration point, 168  
**RemBob()**, 147  
**RemIBob()**, 147  
**Remove()**, 386, 404  
**RemVSprite()**, 125-126  
**Render()**, 431  
**ReplyPort**, 279  
**ReplyPort** pointer, 249  
 RHeight, 19  
 RIGHTHIT flag, 154  
**rightmost** variable  
   in Bobs clipping region, 145  
   in Bob/VSprite collision, 161  
 RINGTRIGGER flag  
   in AnimComps, 173  
   in linking AnimComps, 175  
   moving registration point, 168  
**RingXTrans** variable  
   in ring processing, 175  
   moving registration point, 168  
**RingYTrans** variable  
   in ring processing, 175  
   moving registration point, 168  
 RWidth, 19  
**RxOffset** variable  
   effect on display, 20  
   in **RasInfo** structure, 20  
   in **ViewPort** display memory, 19  
**RyOffset** variable  
   effect on display, 20

- in **RasInfo** structure, 20
- in **ViewPort** display memory, 19
- SAVEBACK** flag
  - in Bobs, 144
  - saving the background, 143
- SAVEBOB** flag
  - changing Bobs, 149
  - in Bobs, 145
- SaveBuffer** variable
  - in saving background, 143
  - with **SAVEBACK**, 144
- SAVEPRESERVE** flag, 146
- scrolling, 51
- ScrollLayer()**, 83, 86
- ScrollRaster()**, 51
- SDCMD\_SETPARAMS**, 385
- SendIO()**, 276
- SER.**, 414
- serial device
  - alternative reading modes, 386-387
  - baud rate, 390
  - bits per read, 391
  - bits per write, 391
  - break commands, 393
  - break conditions, 391
  - buffer size, 390
  - buffers, 385
  - closing, 394
  - end-of-file, 391
  - EOF mode, 388, 391
  - errors, 394
  - exclusive access, 384
  - flags, 384
  - high-speed mode, 391
  - I/O request structures, 385
  - IOExtSer** block, 396
  - io\_TermArray**, 388
  - modes, 383
  - opening timer device, 385
  - parameter changes, 385
  - parity, 393
  - quick I/O, 387
  - reading, 385
  - SDCMD\_SETPARAMS**, 385
  - serial flags, 391
  - seven-wire access, 393
  - seven-wire flag, 384
  - shared access, 384, 391
  - stop bits, 391
  - terminating the read, 388
  - writing, 388
  - xON, xOFF, 390-391
- SetAPen()**, 197
- SetBPen()**, 197
- SetCollision()**, 155
- SetDrPt()**, 44
- SetFont()**, 195
- SetKeyMap()**, 298
- SetRast()**, 50
- SetSoftStyle()**, 199
- SHFCprlist** variable, 33
- short-frame Copper list, 33
- simple refresh, 94
- simple sprites
  - definition, 108
  - GfxBase**, 131
  - in Intuition, 110
  - position, 113
  - routines, 110
- SimpleSprite** structure, 112
- single-buffering, 21
- SizeLayer()**, 82, 86
- software clipping
  - in filling, 46
  - in line drawing, 44
- SortGList()**
  - changing Bobs, 149
  - ordering GEL list, 126
  - sorting Bobs, 148
  - with **DoCollision()**, 177
- sound synthesis, 222
- source** variable, 55
- speech, *see* narrator device
- SprColors** pointer
  - changing VSprites, 129
  - in VSprite troubleshooting, 132
  - in VSprites, 122-123
  - when a 0, 133
- sprFlag** variable, 143
- sprite DMA, 132
- SPRITE** flag, 110
- sprites
  - color, 16, 111
  - display, 13
  - hardware, 108
  - pairs, 111
  - reserving, 131
  - reusability, 108
  - simple, 108
  - virtual, 108
- sprRsrvd** variable
  - effect on Bob color, 153
  - in reserving sprites, 131
- srcMod** variable, 55
- srcX** variable, 55
- stencil drawing, 52
- SubTime()**, 251
- SwapBitsClipRectRastPort()**, 84

- system time, 251
- TD\_CHANGENUM command, 269
- TD\_CHANGESTATE command, 270
- TD\_FORMAT command, 269
- TD\_MOTOR command, 268
- TD\_PROTSTATUS command, 270
- TD\_REMOVE command, 269
- TD\_SEEK command, 270
- text
  - adding fonts, 200
  - baseline, 193
  - changing font style, 199
  - character data, 206
  - color, 197
  - default fonts, 195
  - defining fonts, 203
  - disk fonts, 202
  - font accessors, 205
  - inter-character spacing, 200
  - printing, 194
  - selecting a font, 195
- TextAttr** structure, 195
- TextFont** structure, 203
- Text()**, 194, 199
- text.h*, 195
- ThinLayerInfo()**, 80
- time events, 322
- timer device
  - arithmetic routines, 253
  - OpenDevice()**, 249
  - units, 248
  - with parallel device, 403
  - with serial device, 385
- TimerBase** variable, 253
- timeRequest** structure, 248
- Timer** variable, 174
- TimeSet** variable
  - with **Animate()**, 174
- timeval** structure, 248
- ToolTypes** array, 487
- TOPHIT flag, 154
- topmost** variable
  - in Bobs clipping region, 145
  - in Bob/VSprite collision, 161
- trackdisk device
  - diagnostic commands, 270
  - error codes, 270
  - OpenDevice()**, 266
  - status commands, 269
- Translate()**, 362
- translator library
  - exception table, 363
- TR\_GETSYSTIME, 251
- TR\_GETSYSTIME command, 251
- TR\_SETSYSTIME command, 251
- Unlocklayer()**, 81
- UnlockLayers()**, 81
- UpfrontLayer()**, 83
- user copper lists, 61
- UserExt** variable, 161
- UserStuff** variables, 162
- VBEAM counter, 60
- VBLANK timer unit, 248
- video priority
  - Bobs**, 105
  - in dual-playfield mode, 15
- View** structure
  - Copper lists in, 33
  - function, 10
  - preparing, 22
- ViewPort**
  - colors, 13, 24
  - display instructions, 25
  - display memory, 19
  - displaying, 11
  - function, 10
  - height, 12
  - in screens, 11
  - interlaced, 18
  - low-resolution, 22
  - modes, 14-15
  - multiple, 22
  - parameters, 12
  - width, 13
  - width of and sprite display, 13
  - windows, 11
- ViewPort** structure, 22
- VP\_HIDE flag, 16
- VSOVERFLOW flag
  - reserving sprites, 131
  - with VSprites, 125
- VSPRITE flag
  - in Bobs, 144
  - in VSprites, 125
- VSprites**
  - adding new features, 161
  - building the Copper list, 127
  - changing, 129
  - color, 121
  - colors, 133
  - definition, 108
  - dummy, 106
  - hardware sprite assignment, 126, 132
  - in Intuition, 124
  - merging instructions, 128
  - playfield colors, 134
  - position, 124
  - shape, 122

- size, 121
- sorting the GEL list, 126
- troubleshooting, 132
- turning on the display, 127
- WaitTOF()**, 129
- WhichLayer()**, 83
- Width** variable, 136
- Window** structure, 275
- Workbench
  - info file, 481
  - sample startup program, 489
  - start-up code, 486
  - start-up message, 485-486
  - ToolTypes**, 487
- Workbench object, 480
- WritePixel()**, 43
- XAccel** variable, 168
- xl** variable, 52
- xmax** variable, 48
- xmin** variable, 48
- XorRectRegion()**, 96
- XTrans**, 166
- XVel** variable, 168
- YAccel** variable, 168
- yl** variable, 53
- ymax** variable, 48
- ymin** variable, 48
- YTrans**, 166
- YVel** variable, 168



*Amiga™ Technical Reference Series*  
**Amiga ROM Kernel Reference Manual:  
Libraries and Devices**

The Amiga Computer is an exciting new high-performance microcomputer with superb graphics, sound, and multitasking capabilities. Its technologically advanced hardware, designed around the Motorola 68000 microprocessor, includes three sophisticated custom chips that control graphics, audio, and peripherals. The Amiga's unique system software is contained in 192K of read-only memory (ROM), providing programmers with unparalleled power, flexibility, and convenience in designing and creating programs.

The AMIGA ROM KERNEL REFERENCE MANUAL: *Libraries and Devices*, written by the technical staff at Commodore-Amiga, Inc., is a detailed introduction to and description of the hundreds of graphics, animation, text, math, and audio routines that make up the Amiga's ROM. This book includes:

- an introduction to how libraries and devices are designed and used
- hundreds of examples to illustrate the uses of the ROM routines
- an in-depth tutorial on graphics and animation
- a complete listing of the libraries and devices in Amiga's ROM

For the serious programmer working in assembly language, C, or Pascal who wants to take full advantage of the Amiga's impressive capabilities, the AMIGA ROM KERNEL REFERENCE MANUAL: *Libraries and Devices* is an essential reference.

Written by the technical staff at Commodore-Amiga, Inc., who designed the Amiga's hardware and system software, the AMIGA ROM KERNEL REFERENCE MANUAL: *Libraries and Devices* is the definitive source of information on the libraries and devices built into this revolutionary microcomputer.

The other books in the *Amiga Technical Reference Series* are:

*Amiga Hardware Reference Manual*  
*Amiga Intuition Reference Manual*  
*Amiga ROM Kernel Reference Manual: Exec*

*Cover design by Marshall Henrichs*  
*Cover photograph by Jack Haeger*