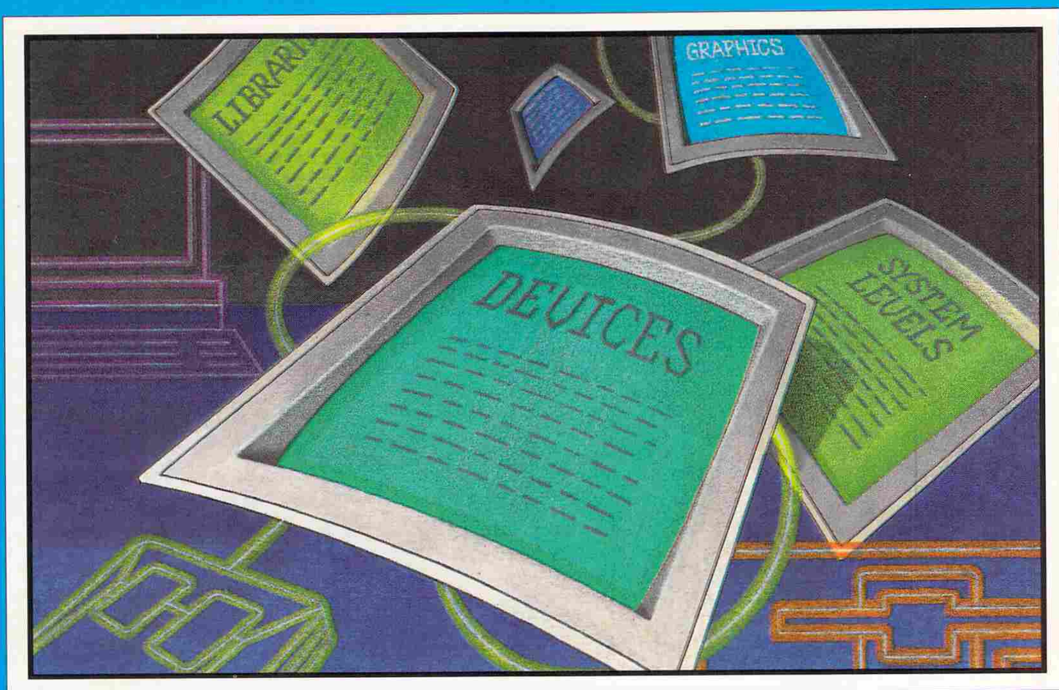
Advanced System Programmer's Guide for the Amiga®

Still more essential information for the Amiga programmer

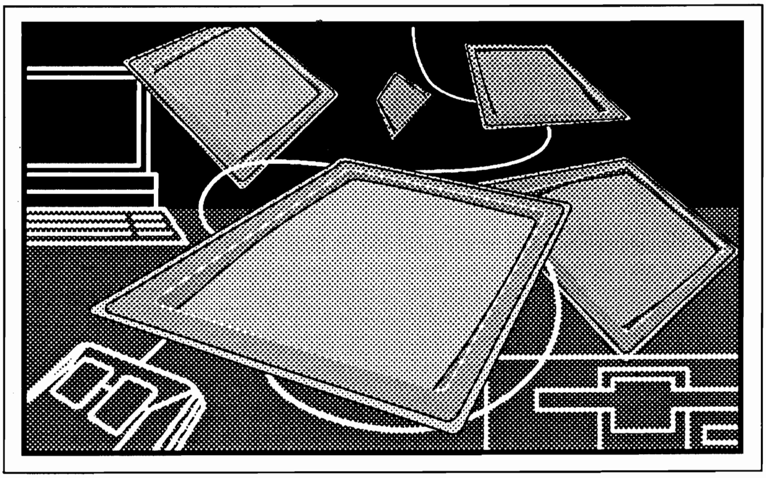


Abacus

A Data Becker Book

Advanced System Programmer's Guide for the Amiga

Bleek
Jennrich
Schulz



A Data Becker Book
Published by
Abacus

First Printing, November 1989
Printed in U.S.A.

Abacus
5370 52nd Street SE
Grand Rapids, MI 49512

Data Becker, GmbH
Merowingerstrasse 30
4000 Duesseldorf, West Germany

ISBN 1-55755-047-6

Foreword

This book is written by programmers for programmers. Although technical books are usually difficult to understand and very practical, this book is different. **Amiga Advanced System Programmers Guide** is similiar to working with the Abacus book **Amiga C for Advanced Programmers.** We have tried to write a book that helps the programmer with programming but doesn't contain too much theory. Our goal was to provide information about large subjects with many example programs.

We wrote this book with more information than any other book on this subject. This book contains complete information about the parameter statements in programs through the CLI and the Workbench, about all of the devices of the Amiga, complete with example programs, about the IFF format from Electronic Arts, and about the basic structures of the Amiga operating system.

Bruno Jennrich spent three months working with all of the devices and thoroughly documented them so that anyone can begin to use them.

Wolf-Gideon Bleek concentrated on the parameter transfer and collected information on programming style. So, you as a programmer can not only program, but also develop a good programming style. He also worked with Intuition and placed all of the information from Preferences together.

Peter Schulz explained the complicated keymaps and math libraries in simple terms. This has made calculations, even with complicated functions, much simpler.

All of the authors contributed to the documentation of all of the libraries and the IFF format, with each writing about the library and the format that he knows best.

You should have a complete collection of information when this book is used in conjunction with a reference book. So this book should always be next to your computer.

W. Bleek, B. Jennrich, P. Schulz

Table of Contents

1. Introduction

Advanced System Programmer's Guide—an impressive title. The name Amiga Advanced System Programmer's Guide tells you exactly what you get: Information about the Amiga operating system, how the system routines function, which of these functions are executable by you, and how the hardware works. This book looks at the Amiga system from an expanded viewpoint.

Chapter 2 of the Amiga Advanced System Programmer's Guide discusses programming style. This chapter builds a base for good style in program development. It thoroughly explains the division, organization and composition of a program. It describes the anatomy of the version number, how to call libraries and how to document your program clearly so that it can be understood by others.

Chapter 3 contains a description of the arguments which appear at the beginning of a program. This explains how to read values that are entered in the CLI, as well as arguments entered through the Workbench. Most commercial Amiga applications use Workbench access instead of the CLI. This chapter also shows which functions are read by the `.info` structures, and shows you how to set the corresponding routine at the beginning of a program. This chapter also shows the two ways to start a program: From the CLI and from the Workbench.

Chapter 4 describes all the Amiga devices available. Devices execute all data exchanges between external or simulated internal devices, this book describes device usage in detail. You'll find explanations for each device's mode and command, as well as demonstration programs. With the help of this book, you can address the Amiga's output and input devices correctly with a minimum of hassle.

Chapter 5 discusses Interchange File Format (IFF), in simple ILBM format as well as the more complex music and text formats. You'll find lists and tables for the music and text formats. Chapter 5 concludes with an explanation of the `Anim` format from Aegis. This represents a complex mixture of ILBM data and the `Anim` format's own chunks.

Chapter 6 comprises the bulk of this Guide. Here you find descriptions of the Amiga library functions, along with documentation for each library. This documentation can be used by both assembly language and C programmers. In addition to general syntax and arguments, each description includes definitions for C variables, which make it easier for the beginner to spot source code errors. Cross referencing makes it possible to quickly find functions dealing with the same subject.

Example: You need information about the function which closes a window, but you can't remember the function's name. You know that it's part of the Intuition library, since Intuition controls all windows. You'd take the following steps:

1. Check the table of contents for the first page of the Intuition library chapter. This chapter contains a listing of Intuition functions.

2. Now look in the group of window functions. Here you'll find `CloseWindow()` and the number of the page describing the function.

3. Turn to the page to find that information.

Rule of thumb: Look in the beginning of the library chapter to find what you need about each function.

Chapter 7 describes the basic structures of the Amiga operating system. These structures are often short but are still needed by every program. Do you use keymaps when you check the keyboard? Do you always have to set Preferences for your program? How is your output formatted when no Amiga fonts are present? The operating system structures take care of all of this and more. This information is presented in demonstration programs.

2. Good Programming Style

We live together on this planet under many rules and conventions. There are conventions of dress, language, and more. Many of these standards are basic, common sense rules. For example, most states consider driving through a red light illegal.

Communicating with a computer must be done under certain rules as well. There are more obvious rules you should follow in programming (e.g., don't hit the Amiga when the system crashes). However, some smaller conventions are very valuable to the user and developer alike.

Think about the Amiga's graphic user interface, Intuition. Certain icons represent certain tools (applications) and projects (files run from specific tools). Some icon designs have become standards, and it's best to keep the standard icon design (e.g., AmigaBASIC projects).

This chapter looks at some helpful rules that you can follow in writing elegant, stylish source code, as well as some hints for better program development.

2.1 Comments and formatting

Comments in source code plays a special role in the field of professional programming. Imagine a company that develops and markets several application programs for the Amiga. After several months Commodore announces a new improved Amiga operating system—which is incompatible with all this company's Amiga applications. Unfortunately, the company's original developer forgot to place comments in the source code, and he now works for another firm. This means that revisions to the program will take much longer than they would have taken with commented code. No one except the original developer knows where anything is in the program, or how crucial routines work.

The same thing can happen to you. Say you develop a large program and put no comments into the source code. You set the program aside for a few weeks and come back to it later. Without comments you have no idea of what works when in the program. The only solution is to analyze the program from the beginning and figure out what it does.

Comment programs carefully, for your own information as well as for others who might study the program later.

Program header

You'll usually find information about the program's name, purpose, author, last date of revision and other information in the program header. Look at the following example, which was taken from the Abacus book **Amiga C for Advanced Programmers**:

```
/****************************************
 *                                      *
 * Program: Window local definition     *
 * ==================================== *
 *                                      *
 * Author:  Date:       Comments:       *
 * ------   ----------  ----------      *
 * Wgb      10/16/1987  first test      *
 *                      window          *
 *                                      *
 ****************************************/
```

The header starts with a brief description of the program. This C program defines a window locally, which means that the data belongs to one function instead of more than one function. The header also lists the author's initials and the date when he/she wrote the program. This program header can be expanded by the author, or by someone who revised this program for his or her own needs.

The program header is the easiest way to ensure that programmers know of any changes, improvements or deletions that have been made to this program. Program headers can contain much more vital information. Here's another program header, taken from the Abacus book **Amiga Tricks & Tips**:

```
'*********************************
'*                               *
'* Open window through Intuition *
'* ----------------------------- *
'*                               *
'* Author  : Wolf-Gideon Bleek   *
'* Date    : May 22, 1988        *
'* Greetings: Denis "Angle"      *
'* Version  : 1.1                *
'* Operating system: V1.2 & V1.3 *
'*                               *
'*********************************
```

Here we also find information about the program's version and the operating system versions. Both values have different meanings. The operating system version may dictate how the program must be compiled. The values are important for BASIC programs because the libraries must be loaded during program execution.

The greeting entry is optional; it can add a personal touch to the code. If you wish to thank someone for their efforts in helping write a program, the program header is the place to insert the note of thanks.

The version number of the program should be in the program text so you can differentiate easily between two similar versions of a program by looking at the version number. This number should also appear in the program itself for the same reason. In other words, the version number should appear twice: Once in the program header, and once in the program itself so that the user can easily find it.

You should specify the operating system version. This is the only way error free use can be guaranteed for the user. You can also use this to recognize if the program uses features that are implemented with the new version. For example, a program which is compatible with Kickstart 1.1 cannot support an auto boot from the hard disk since KickStart 1.1 does not support this feature. On the other hand, a program will only be compatible with every Amiga if it correctly accesses the operating system functions.

More about version numbers

Version numbers follow certain conventions in Amiga development. A version number helps the user and developer quickly recognize the current version of the program. Most version numbers appear as two digits, separated from one another by a decimal point:

```
Version X.Y
```

The number to the left of the decimal point represents the major version. This number is a zero if the program is still in the development phase (e.g., Version 0.1 for an early design). The first fully functional program version appears as version 1.0.

The Y parameter represents the minor version number. The Y usually begins at zero (e.g., version 1.0) and goes up to nine (e.g., version 1.9). The .Y parameter can also be notated in tenths or hundredths. These smaller notations indicate lesser changes to a program such as minor error corrections.

It is up to the programer to change the version number when updating the program. The programmer must decide the extent of the version number change. For example, imagine that you're developing a program. You would assign the first testing version a number of 0.1 (no preceding version numbers exist). If you correct existing errors without adding any new real data, the version numbers can increase from 0.11 to 0.19 (note the added decimal place). If you add a new function to a program, the version number could then be changed to version 0.2.

Let's assume that version 0.7 of your program is completely executable and needs no more expansion. Then you can change the version number to 1.0. The Amiga operating system is a good example of version number changes. Version 1.0 was the first executable version of that system, but required major revisions because of errors found after its release. The next version (1.1) contained these changes. Revisions by the European divisions of Commodore-Amiga resulted in version 1.2. Version 1.3 contained many more improvements. The version in development (1.4) includes special graphic chip support.

Version numbers don't usually go beyond the second or third decimal place. It would be inhuman and impractical to assign the main version of a program a version number of 1.279. This could result in a program package stating the following: "This program runs with all operating system versions of 1.2623 and up, but not with intermediate versions 1.2758 and 1.296." That's why a final version exists—to help users to differentiate between versions easily.

You can find the current Kickstart and Workbench versions by selecting the Version item from the Workbench menu. Version 1.2 of Kickstart displays the number 33.180. This number indicates the library version in use: 33 is the library version number. Later in this chapter you will read about how this can affect the libraries. The Workbench uses version number 33.57, but Kickstart 1.3 increases this version number to 34.7.

The function header

Function headers list information about the function which follows it. The information here is different from that contained in the program header. The function header describes syntax, parameters that are transferred or returned and variables that are changed by a subroutine. Here is an example:

```
/***************************************
*                                      *
*  Function to erase an already        *
*  existing Lexicon structure:         *
*  FreeLexicon()                       *
*                                      *
*  Additions:                          *
*  - support of entries                *
*  - select, if saved                  *
*                                      *
*  Input parameters:                   *
*  Lexicon - pointer to best structure *
*                                      *
*  Return parameters:                  *
*  none                                *
*                                      *
*  Date:  April 3,  1988               *
*  Author: Wolf-Gideon Bleek           *
*  Greetings: Christian                *
*                                      *
***************************************/
```

The first lines of text in the function header clearly define the function's purpose. The third line of text states the C function's name. These items are most important to the majority of programmers.

The Additions lines prove to be very useful in the development phases of a function. Changes may be started during a programming session and not completed at the end of that particular session. These additions tell the programmer where development left off for the next session.

The input parameters specify any values needed by the function. C, AmigaBASIC and other languages allow long parameter names. We recommend that you use names that clearly describe the variable whenever possible.

The return parameters describe the results of this routine. Functions often have only one return parameter, if any. If a parameter name is required, use names that clearly describe the variable.

The example above includes the date of completion, the author's name and a personal greeting. This can be expanded as needed.

2.2 Development and multitasking

There are several rules that should be kept in mind when designing your program to work with a multitasking operating system. These rules ensure that your program does not conflict with other programs. Opening and closing communication channels causes the most problems. This refers to communication with the user, the devices and the system routines. Management systems regulate these devices—no direct access takes place.

A communication channel must be opened before communication takes place. This is similar to a door leading to a room: You can't exchange information with people in the next room if the door is closed. Once you open the door, you can communicate with anyone in the adjacent room. When you close the door, communication ends.

One rule suggests that you close a door when you don't need to enter a room. The same goes for the computer: Close all of the data channels when you no longer need them.

The telephone offers another illustration of communication channels. When you want to call someone, you pick up the receiver and dial a number. One of two things happens: Either you make the connection and the telephone on the other end starts ringing, or you hear a busy signal. The only option when a busy signal occurs is to hang up the telephone and dial again. The Amiga also returns a busy signal if it cannot currently open a library or something similar. One of two things occurred: Either there was no memory available, or the channel was non-existent.

Let's look at the processes used for opening and closing libraries, devices or data channels. We need a routine which opens a library. All it requires is the library's name and version number. When we try to open a library, the routine either returns the base address of the library or a zero (=error). The routine must constantly check for errors, since the possibility of errors always exists. The following routine performs this error check:

```
Library = OpenLibrary("LibName", Version);
IF (Library == Null)
   Cancel();
```

Version specifies the version number that your program needs. You can find the version number by entering the AmigaDOS Version command from the CLI. Stating a Version value of zero allows program access to all the libraries, even those not normally accessible through command words. The following line combines the opening and the error check into one short routine:

```
IF (!(Library = OpenLibrary("LibName", Version)))
  exit(FALSE);
```

Closing a library is very simple. You must never try to close an unopened library, otherwise a system crash occurs. The following checks a pointer which ensures that the library can be closed:

```
IF (Library) CloseLibrary(Library);
```

The closing procedure becomes much more complicated if a program must execute multiple file accesses. All of the open libraries must be closed. We need a routine which can find all open libraries and close the open libraries only.

Let's look at an example. Your program needs two system libraries, a window and multiple blocks of memory. The program displays an error message when you try to open the window. A poor programmer would simply stop the program at this point, but this creates problems for the multitasking system. Remember that the two libraries reserved for the program take up memory. Tasks running at the same time slow down because of low available memory and active library access.

We can limit memory loss and open access using the `Open_All()` and `Close_All()` functions listed in the program below. These functions open and close the necessary channels as needed. When an error occurs while opening any area of memory, the program jumps to the `Close_All` routine and ends the program. The `Close_All` routine knows what is open and what is not. The following demonstrates how this works:

```
/******************************************
 * Function: Open Libraries and Window    *
 * ====================================== *
 *                                        *
 * Author:  Date:      Comments:          *
 * ------  ----------  ----------         *
 * Wgb     06/15/1988  also memory        *
 *                                        *
 ******************************************/
Open_All()
   {
   void          *OpenLibrary();
   struct Window *OpenWindow();

   if (!(IntuitionBase = (struct IntuitionBase *)
       OpenLibrary("intuition.library", 0L)))
      {
```

```
      printf("No Intuition library found!\n");
      Close_All();
      exit(FALSE);
      }
   if (!(GfxBase = (struct GfxBase *)
       OpenLibrary("graphics.library", 0L)))
      {
      printf("No Graphics library found!\n");
      Close_All();
      exit(FALSE);
      }
   if (!(FirstWindow = (struct Window *)
       OpenWindow(&FirstNewWindow)))
      {
      printf("Window will not open!\n");
      Close_All();
      exit(FALSE);
      }

   UndoBuffer = AllocMem(512L, MemoryType);
   if (!UndoBuffer)
      {
      printf("Problems with Undo buffer!\n");
      Close_All();
      exit(FALSE);
      }

   FileBuffer = AllocMem(30L, MemoryType);
   if (!FileBuffer)
      {
      printf("Problems with File buffer!\n");
      Close_All();
      exit(FALSE);
      }
   }

/****************************************
 * Function:Close everything that is open*
 * ==================================   *
 *                                       *
 * Author:  Date:      Comments:         *
 * ------  ----------  ----------        *
 * Wgb     15.06.1988  Intuition, Window *
 *                     Graphics & Mem    *
 ****************************************/
Close_All()
   {
   if (FirstWindow)     CloseWindow(FirstWindow);
   if (IntuitionBase)   CloseLibrary(IntuitionBase);
   if (GfxBase)         CloseLibrary(GfxBase);
   if (UndoBuffer)      FreeMem(UndoBuffer, 512L);
   if (FileBuffer)      FreeMem(FileBuffer, 30L);
   }
```

The `Close_All()` function fulfills all of the requirements that we set for ourselves. It can be accessed from any place in the program, and it closes anything that it knows is open. Anything unopened is ignored, decreasing the odds of fatal errors.

The use of the functions as listed here is tightly structured. You may want to move all `Open_All()` calls to the beginning of the program to save some program memory.

You could place the `Close_All()` routine at the end of the `Main()` function, but that would require a `goto` whenever an error occurs. This is not highly structured C programming, but it is a legitimate and practical solution. In addition, placing `Close_All()` after `Main()` saves a few bytes of program memory.

2.3 Assembly language programming

Assembly language offers much more freedom in programming than the C language. Assembly language code can be executed directly, without the compiler and linker needed by C source codes. Higher level languages and assembly language must be able to handle certain services called registers. The Amiga's microprocessor has eight data registers and seven address registers in addition to the address counter, the two stack pointers and the status register.

All of these registers can be accessed from a program. However, there are some rules that you must heed if you want to correctly use the Amiga's operating system. Each library function assumes that you know which register is used and which registers remain unused.

Register Tabula rasa ("forbidden registers"):

The first forbidden register is in address register A7. It usually acts as the stack pointer (SP), and cannot be loaded with its own data. Like SP, you cannot load the user stack pointer (USP) and the supervisor stack pointer (SSP) with their own data. Changing these three registers requires a high level of system knowledge, and even then any changes to these registers should be avoided. Changing these registers disables multitasking, which defeats one of the Amiga's biggest features.

Using registers in conjunction with libraries

Several factors must be considered when using the library routines in a program. First and foremost is specifying the base address register. Calling the library routine places the base address of the library in register A6. This address could then be passed to another register. The library routines also use internal variables that are addressed through this register. Each function assumes that it finds its base address in register A6. You can also execute a call that always stands at the base value.

Next, look at the first two address and data registers (A0, A1 & D0, D1). Almost every function uses these registers at some time or another. The contents of these four registers are neither saved nor restored. As you program, remember that these registers do not contain important data throughout the entire program, or even during a longer function. These registers work best for small data transfers and calculations.

The dos.library uses the D2 and D3 registers constantly because of a constant need for buffer memory. Keep these registers in mind when accessing the dos.library from a program. In addition, notice that all of the other registers called by the function are saved.

We wish to comment here about saving of registers. When you write your own functions in your programs, state in the function header which registers are saved and which are not saved. This makes it much easier in later development. Hold to the library conventions of register listing to avoid confusion.

Note: Save registers while within a function—not before the function call. Clearly format these register saves, and never try to cross registers.

2.4 Accessing system directories

Many programs on the Amiga won't run without system support. This support comes from data already in the system (fonts, printer drivers, libraries, etc.). Where do you find the data when you want to use it?

The system directories contain most of the additional files and programs needed for system support. These directories always exist under one general name to minimize the time spent by the program searching for system files.

Let's look at the existing directories. If you enter the `Assign` command from the CLI the following list appears (your list may look slightly different from this one, since our Workbench disk is named Wgb):

```
Directorys:
ENV            RAM DISK:Env
T              RAM DISK:T
FONTS          Workbench 1.3 Wgb:fonts
S              Workbench 1.3 Wgb:S
L              Workbench 1.3 Wgb:L
C              Workbench 1.3 Wgb:c
DEVS           Workbench 1.3 Wgb:devs
LIBS           Workbench 1.3 Wgb:libs
SYS            Workbench 1.3 Wgb:
```

The directories in this list can be addressed under a generally known name. This general name carries through each directory to the actual device and its directory.

Here's an example. You've developed a program that sends any text to a printer. The user can select an alternate Amiga font. The program takes this font from the Workbench diskette in drive DF0:. The syntax would look something like this:

```
DF0:fonts/name
```

This command sequence raises some questions:

- The DF0: tells the system to look in the internal disk drive. What about users who have two disk drives, and place the Workbench disk in an external drive?
- How can the user tell the program to look on the hard disk for fonts?

- The program will look specifically for the fonts directory. What if the user wants to change the name of the fonts directory to something else and still have the program access it?
- All fonts must be on the Workbench disk. What about the user who wants a font not in the fonts directory?

You can see that this solution is not ideal. The Amiga's operating system accesses the directory with the desired contents through the abovementioned name. The `Assign` command allows you to assign the fonts directory, but the program will still access the fonts directory, whatever drive this directory is on.

The developer should know what exists in which directory. The following list describes the contents of each system directory.

FONTS

Workbench 1.3 Wgb:fonts:

The fonts directory contains all the available fonts. This directory is primarily addressed by the diskfont.library. You can add new fonts or delete existing fonts.

S

Workbench 1.3 Wgb:s:

The S directory contains script files. Versions of the operating system up to and including 1.2 contain only the script file named startup-sequence. Version 1.3 includes a second startup-sequence for the CLI and the Shell, as well as a demo version of the HardDisk startup sequence. This directory is useful because the AmigaDOS Execute command searches this directory for script files if they cannot be found in the current path. This saves the user some typing, and keeps all of the script files in one directory. The system searches the S directory during booting and executes the file named startup-sequence.

L

Workbench 1.3 Wgb:l:

The L directory contains the handlers for all of the non-resident libraries. A new library can be included using the `OverlayCodeSegments`.

C

Workbench 1.3 Wgb:c:

The C directory contains all of the CLI commands. As soon as a command is entered from a Shell or CLI window, the CLI searches this directory for the command. The command list can be expanded using a path.

DEVS **Workbench 1.3 Wgb:devs:**

The devs directory contains many devices. Not all of the Amiga's devices are implemented in the operating system, since some are only used infrequently. The operating system searches the devs directory after an `OpenDevice()` call if the device is not already present in memory. You can write your own devices and store them here if desired.

LIBS **Workbench 1.3 Wgb:libs:**

The libs directory contains many libraries. You can write your own libraries and store them here if desired.

SYS **Workbench 1.3 Wgb:**

SYS represents the label on the system disk used for booting. The user can access this disk by entering SYS: instead of the drive specifier. This is especially useful when you must change your Workbench disk from drive DF0 to another drive. When you enter SYS: from the CLI or Shell, the Amiga automatically changes Workbench disk access.

T Many applications access the T directory, which is used for storing backup and temporary files. Version 1.3 of the operating system renames the device in T: so that all of the advantages that we have learned about can be used. When developing a program never place this directory in the RAM disk. Even though access is faster in a RAM disk, all of the information is lost after a reset.

2.5 Intuition

A big factor in program development lies in writing for the end user. Intuition offers us an alternative to using the Shell and CLI for access.

Menus

Menus allow the user to select different items. We're going to look at the menu display for now, instead of programming menus. No matter how helpful the menus may be, poor presentation can ruin the menus. Here are some rules to help you when developing menu routines.

Write out a list of all the menu titles and items you want to include in the program. The menu line can contain up to ten menu titles, which appear on the menu line when you press the right mouse key. Choose menu titles carefully—make sure that these titles clearly explain the items below the titles.

The menu titles should be placed starting with the most important (or most used) title farthest to the left. The titles should decrease in importance from left to right. Most often the left menu title is the File menu title.

Shortcuts

Include keyboard shortcuts for frequently used menu items (e.g., <Amiga><s> for Save from the File menu). Remember that the Amiga is case sensitive (i.e., <Amiga><s> and <Amiga><S> can represent two separate shortcuts). Special characters are also allowed.

Menu items should be placed in the most logical order possible. For example, a File menu should display its New, Load and Save items first, followed by items for file deletion and other file maintenance.

If more than one option is available for an item (e.g., Save and Save As), a submenu or requester can be added. For example, if the program offers a choice of formats in which the data can be saved (ASCII, IFF, protected format), you can add a submenu listing the ASCII, IFF and PROTECTED items to the Save item. This eliminates the need for a requester and saves you some programming time.

Display a requester as a last warning to the user that something important is about to happen (e.g., deleting a file).

Requesters

Requesters are smaller windows which request information of confirmation from the user. Requesters can contain gadgets displaying words such as YES, OK and CANCEL, or string gadgets into which the user can type text.

Sketch out the way you want the requester to appear. The requester must appear organized and clear. For example, a requester appearing to confirm execution of an important function (e.g., deleting a file) should contain at least two gadgets to allow positive or negative feedback from the user. A simple YES and NO gadget will serve this purpose, or even OK and CANCEL.

Include a gadget for alternate selection in a requester. For the abovementioned requester used to confirm file deletion, you could have a YES gadget, a NO gadget and even a BACKUP ONLY gadget to allow the user to delete only the backup copy of the file.

Key response Some gadgets let the user select a gadget from the keyboard instead of the mouse. The keyboard is not supported from Intuition like the menus. Intuition handles the keyboard as a gadget specifically assigned to one of the gadgets in the requester.For example, **BeckerText** from Abacus surrounds this gadget with a bold red border. When the user presses the <Return> key while in an active requester, the reading routine reads the <Return> key as the specified gadget.

Choose this keyboard actuated gadget carefully (e.g., do not make the YES gadget of a Delete file requester accessible from the keyboard).

3. Data Transfer

Sometimes user interaction with a program begins before the program even runs. This early interaction involves the entry of arguments (parameters), which may not take effect immediately. For example, if you enter the following from the CLI, the text editor ED loads into memory, then a file loads into ED and appears on the screen:

```
ed filename
```

This same process can be used from the Workbench. You can move the mouse pointer onto the icon of a text file created by a word processor (e.g., **BeckerText**) and double-click on the text file icon. The word processor loads first (assuming it is readily available), then the operating system loads the selected file into the word processor.

Getting user data

The programmer can use these two methods to get data from the user. Some programs cannot operate without additional arguments. Many Shell and CLI commands require arguments to operate correctly.

We've just seen how some programs require certain data to execute correctly. This chapter views the transfer of data from two sources. First we'll examine how the CLI accepts arguments from the user when invoking a command. C programming for arguments is very simple, thanks to the way the C compiler processes data. Then we'll see how data is transferred by the Workbench.

3.1 CLI arguments

Many CLI commands require additional arguments. The syntax can look something like this:

```
cli_command argument
```

The invoked CLI command reads and confirms the arguments, then executes the command based on these arguments if possible.

Arguments

The `main()` function of the C language has two arguments that are seldom used but are very helpful. The first argument specifies the number of arguments needed. The second argument represents a pointer to a string array into which all of the arguments are placed. The following routine reads the input line, from which we can read the individual entries.

```
/***************************************
 *                                     *
 * Subroutine: Read the input line     *
 * =================================== *
 * Inputsub.c                          *
 * Author:  Date:       Comments:      *
 * ------  ----------  ----------      *
 * Wgb     July 1988   Aztec Routine   *
 *                                     *
 *                                     *
 ***************************************/
#include <libraries/dosextens.h>
extern int _argc, _arg_len;
extern char **_argv, *_arg_lin;
_cli_parse(pp, alen, aptr)
struct Process *pp;
long alen;
register char *aptr;
   {
   register char *cp;        /* Character Pointer */
   register struct CommandLineInterface *cli;
   register int c;           /* Characters at Pointer Position */
   void *_AllocMem();
   cli= (struct CommandLineInterface *) ((long)pp->pr_CLI << 2);
   cp  = (char *)((long)cli->cli_CommandName << 2);
   _arg_len = cp[0]+alen+2; /* Length + PrgName + Null-Bytes */
   if ((_arg_lin = _AllocMem((long)_arg_len, 0L)) == 0)
      return;
   strncpy(_arg_lin, cp+1, cp[0]);     /* Program name */
   strcpy(_arg_lin+cp[0], " ");        /* spaces  */
   strncat(_arg_lin, aptr, (int)alen); /* Parameter    */
   for (_argc=0,aptr=cp=_arg_lin;;_argc++)
      {
      while ((c=*cp) == ' ' || c == '\t' || c == '\f' ||
               c == '\r' || c == '\n')
```

```
            cp++;
         if (*cp < ' ')
            break;
         if (*cp == '"')
            {
            cp++;
            while (c = *cp++)
               {
               *aptr++ = c;
               if (c == '"')
                  {
                  if (*cp == '"')
                     cp++;
                  else
                     {
                     aptr[-1] = 0;
                     break;
                     }
                  }
               }
            }
         else
            {
            while((c=*cp++) && c != ' ' && c != '\t' && c!= '\f' &&
                       c != '\r' && c != '\n')
               *aptr++ = c;
            *aptr++ = 0;
            }
         if (c == 0)
            --cp;
         }
      *aptr = 0;
      if((_argv= _AllocMem((long)(_argc+1)*sizeof(*_argv),0L))== 0)
         {
         _argc = 0;
         return;
         }
      for (c=0,cp=_arg_lin;c<_argc;c++)
         {
         _argv[c] = cp;
         cp += strlen(cp) + 1;
         }
      _argv[c] = 0;
      }
```

Program description

The program takes the length of the input line from the CLI, then allocates memory for the argument tables. When these are not present, it means that no arguments were given in the command line. The program name, spaces and arguments are then copied into the argument table memory. This serves as the base for the following loop.

The loop checks the entire list for any separator characters (i.e., spaces, tabs, form feeds, carriage returns and ends of lines). When the loop finds one of these characters, it determines how the characters are handled, and whether a command character is found. If one of the command characters is encountered, it is determined in many comparisons which characters are handled or if a quotation mark is encountered. This quotation mark tells the loop to look after the quotation mark for additional separators.

Quotation marks

After this test, the loop treats the character string between the quotes as if no separating characters are found. Then the program loop places a null byte after the text to end it. This operation repeats until the entire text has been checked. Then another memory function allocates new memory for the pointer tables (more on this later). All of the pointers to the individual texts are entered in this block of memory.

The last entry in the pointer table is set to zero to make the end recognizable without number variables. To see how this routine works, here is a program that displays the number of parameters given as CLI arguments, then lists all of the values in table form.

```
/****************************************
 *                                      *
 * Program: Display CLI arguments       *
 * ================================== *
 *  DisplayCli.c                        *
 * Author:  Date:      Comments:        *
 * ------   ----------  ----------      *
 * Wgb      June.1988   only listing    *
 ****************************************/
main(ArgC, ArgV)
int ArgC;
char *ArgV[];
   {
   int i;

   printf("Number of arguments: %d\n", ArgC);

   for (i=0; i<ArgC; i++)
   printf("CLIArg %d: >%s<\n", i, ArgV[i]);
   }
```

Program description

The `main()` function handles the `ArgC` and `ArgV` arguments. `ArgC` (Argument Counter) represents a counter variable which contains the total number of assigned arguments. The program name is considered one of these arguments within the entire input line. This input line is assigned to a text table. The program name can include a disk path, if such a path is needed. Program names are included as arguments, even when a program is started from the Workbench (more on this later).

`ArgV` (Argument Vector) is the pointer to the text table mentioned previously. It is handled as a one-dimensional array of `char` elements (characters). The routine created from the entries of this table recognizes spaces between two words as a break between two arguments. There may be times when a space is required in a text (e.g., `file name` instead of `filename`). If you wish to read two words as one argument, then the desired text must be placed within quotation marks. For example:

Wrong: `execute file name`
Right: `execute "file name"`

This program is intended to show you a simple application of the routine. Enter the program and save it under the name `DisplayCLI.C`. Compile and link normally according to your compiler's instruction manual (e.g., Aztec C uses the 32 bit optio).

Let's execute the program a few times to test it out. Enter the CLI and enter the following (add your own path names as needed):

```
DisplayCLI
```

The program displays the following on the screen:

```
Number of arguments: 1
CLIArg 0: <DisplayCli>
```

Let's execute the program with some arguments. Enter the following:

```
DisplayCLI 1. DF0: Hello Wally
```

The program displays the following on the screen:

```
Number of arguments: 5
CLIArg 0: <DisplayCLI>
CLIArg 1: <1>.
CLIArg 2: <DF0:>
CLIArg 3: <Hello>
CLIArg 4: <Wally>
```

More about quotation marks

Earlier we discussed adding quotation marks to make multiple words into one argument. Let's see if the program accepts this type of input. Enter the following:

```
DisplayCLI "DF1:1. Test Run"
```

The program displays the following on the screen:

```
Number of arguments: 2
CLIArg  0: <DisplayCLI>
CLIArg  1: <DF1:1. Test Run>
```

Argument templates

Let's take a closer look at arguments. Programs should always have an *argument template* available. An argument template tells the user which arguments are acceptable to a command. You can display an argument template from the CLI by entering the command name, a space, a question mark and the <Return> key.

AmigaDOS has argument templates available for almost all of its commands. For example, if you wanted to see the argument template for the `dir` command, you would enter the following in the CLI command line:

```
dir ?<Return>
```

Testing for argument templates

Our own program should have the ability to test for a question mark. It should also test for the proper number of arguments. If it detects an incorrect number of arguments, the program should display an appropriate error message.

The following program executes this task. We made this the `main()` function of the program. You may wish to follow this style in your own programming.

```
/****************************************
 * Program: Read CLI arguments          *
 * ==================================== *
 * ReadCLI.c                            *
 * Author:  Date:       Comments:       *
 * ------   ----------  ----------      *
 * Wgb      06/20/1988  also "?" Option *
 ****************************************/
#include <exec/types.h>
main(ArgC, ArgV)
int ArgC;
UBYTE *ArgV[];
   {
   int i;
   if (ArgC == 1)
      printf("No parameters from the CLI!\n");
   else
      printf("%d Parameter, that can be evaluated\n", ArgC-1);
   if ((ArgC == 2) && (*ArgV[1] == '?'))
      printf("Format: %s [...] [...]\n", ArgV[0]);
   }
```

Program description

The program displays the number of arguments given. If no arguments were entered, the program displays this fact. The program also checks for <Space><?>. If the user entered the request for the argument template, the program displays the argument template on the screen.

The argument template may take up more than one screen line. Our program reads the first entry in the text table, rather than the program name itself. This allows the user to rename programs and assign different paths as needed, thus keeping the program open to change.

Remember that the above program listing shows only a few aspects of argument template output. You've probably seen programs that react to too few arguments, too many arguments, or even incorrect arguments. The single disadvantage of the " ?" function is that the entire program must be loaded before the argument template can be displayed. Sometimes the argument template may not appear because of insufficient memory.

3.2 The .info file

The DisplayCLI program listed above demonstrated a method of making the CLI easier to use. However, programmers and developers are the most frequent users of the CLI. The average user accesses programs through the Workbench. This user interface ensures simple access for the user who just wants to use the computer with a minimum of computer literacy.

Arguments and the Workbench

How can the user enter arguments in an application started from the Workbench? Let's look at what happens when we start a program from Workbench. The program receives information from a startup routine determined by the main() function. This startup routine replaces the argument table given by text entered in the CLI, while assigning values from the Workbench. We'll now look more closely at these values.

The first data received is a list of files and locks available to this program. The files refer to a list of files invoked when you double-click a program icon. The locks are pointers to the directories, supplying the program with pure filenames. Look at the following program:

```
/****************************************
 *                                      *
 * Program: List out WBMessage          *
 * ==================================== *
 *  WBMessage.c                         *
 * Author:  Date:       Comments:       *
 * ------  ----------   ----------      *
 * Wgb     06/20/1988   only Locks and  *
 *                      File names      *
 *                                      *
 ****************************************/
#include <exec/types.h>
#include <workbench/startup.h>
#include <stdio.h>
extern struct WBStartup *WBenchMsg;
main()
   {
   int i;
   struct WBArg *Arg;

   for (i=0, Arg=WBenchMsg->sm_ArgList; i<WBenchMsg->sm_NumArgs;
        i++, Arg++)
      {
      printf("WBArg %d: Lock=0x%lx Name = %s\n",
              i, Arg->wa_Lock, Arg->wa_Name);
      }
   printf("PRESS <Return> TO EXIT\n");
   Delay(5*60L);
   }
```

Program description

This program assumes that it was started from the Workbench. Later you will see an example of a program that can be started either from the CLI or the Workbench. It displays the message `WBenchMsg`, which contains all of the applicable filenames and corresponding locks. These are listed in table format in the `for()` loop.

Compile and link the program, then assign it a tool icon. Once you've done that, we can begin experimenting. Open the Workbench screen and select the icon of this program. Select the Workbench function `Info` item from the `Workbench` menu. The `ToolTypes` string gadget should contain the following:

```
WINDOW=CON:0/0/600/80/TestWindow
```

Immediately after starting the program, the startup routine mentioned automatically opens the window as defined in the `ToolTypes` string gadget. The short routine listed below performs this task:

```
/*******************************************
 *                                         *
 * Program: Read ToolTypes in window       *
 * ====================================== *
 * ReadToolTypes.c                         *
 * Author:  Date:       Comments:          *
 * ------  ----------  ----------          *
 * Wgb     July 1988   Aztecs Routine      *
 *                                         *
 *                                         *
 *******************************************/
#include <libraries/dosextens.h>
#include <workbench/workbench.h>
#include <workbench/startup.h>
#include <workbench/icon.h>
void *IconBase = 0;
_wb_parse(pp, wbm)
register struct Process *pp;
struct WBStartup *wbm;
   {
   register char *cp;
   register struct DiskObject *dop;
   register struct FileHandle *fhp;
   register long wind;
   void *_OpenLibrary();
   long _Open();
   if ((IconBase = _OpenLibrary("icon.library", 0L)) == 0)
      return;
   if ((dop = GetDiskObject(wbm->sm_ArgList->wa_Name)) == 0)
      goto closeit;
   if (cp = FindToolType(dop->do_ToolTypes, "WINDOW"))
      {
      if (wind = _Open(cp, MODE_OLDFILE))
         {
         fhp = (struct FileHandle *) (wind << 2);
         pp->pr_ConsoleTask = (APTR) fhp->fh_Type;
         pp->pr_CIS         = (BPTR) wind;
         pp->pr_COS         = (BPTR) _Open("*", MODE_OLDFILE);
         }
```

```
      }
   FreeDiskObject(dop);
closeit:
   CloseLibrary(IconBase);
   IconBase = 0;
   }
```

Program description

This routine tries to open the `Icon.library` to determine whether a ToolType WINDOW appears in the program file. Then the .info structure is accessed by means of `GetDiskObject()`, and `FindToolType` examines the structure. If an entry named WINDOW is present, the routine attempts to open an output window using the definition following WINDOW. If this is successful, the window opens and the routine informs the structure that execution was successful.

After opening the window, `DiskObject` releases the library. We click on the program icon and our program starts. The window appears in the predefined location and displays an entry from the table. We also see the directory lock, which contains the program and the program name.

You can access more than one .info file from the Workbench, just as you can call multiple programs in the CLI. Click on another icon, press the <Shift> key and double-click the program's icon: Another entry appears in the list.

Let's examine another way to do this. Create a Notepad text and copy it to the disk which also contains `WBMessage`. Select the `Info` item from the `Workbench` menu. When the Info screen appears, look at the string gadget labeled Default tool. It should contain the text:

```
SYS:Utilities/Notepad
```

Click on this gadget and delete this text. Enter the following in its place:

```
DF0:WBMessage
```

Click on the Save gadget to exit. Now double-click on the Notepad text. The window reappears. This time, the Notepad text tried to access a non-existent Default Tool (i.e., WBMessage). The lock entry displays no value, and the program lists the name of this tool as it appears in the `Default Tool` string gadget. We find the lock and the corresponding name under the text entry.

More about Tools and Projects

We can also determine whether a Tool (an application) was started directly or through a Project icon. Add the following data to the above program code:

```
 printf("WBArg %d: Lock=0x%lx Name = %s\n",
```

```
        i, Arg->wa_Lock, Arg->wa_Name);
 if ((i == 0) && (Arg->wa_Lock == 0))
    {
    printf("Started without program. This was loaded
afterwards!\n");
    }
```

Compile and link the source, then assign it an icon. Click on the Notepad text you created in the last example and select the `Duplicate` item from the `Workbench` menu. Press the <Shift> key and click on both Notepad text icons. Now double-click the program icon. We get a longer list with more locks and filenames. What sense does it make to use all of the filenames? Each filename looks to the data in the .info file. We can read this data, examine it and even process it.

The `GetDiskObject()` function, which lies within the `icon.library`, reads the data from the .info file. We can set a new current directory using the lock, then read the .info file:. Look at the following program code:

```
/****************************************
 *                                      *
 * Program: List out ToolTypes          *
 * ==================================== *
 * ListToolTypes.C                      *
 * Author:  Date:      Comments:        *
 * ------  ----------  ----------       *
 * Wgb     06/20/1988  Access to .info  *
 *                     file only        *
 * Compile options                      *
 * cc +L listtooltypes.c                *
 * ln listtooltypes.o -lc32             *
 ****************************************/
#include <exec/types.h>
#include <workbench/workbench.h>
#include <workbench/startup.h>
#include <workbench/icon.h>
#include <stdio.h>
extern struct WBStartup *WBenchMsg;
extern struct IconBase *IconBase;
void                    *OpenLibrary();
main()
   {
   int i, j;
   char **ToolArray, *Value;
   LONG OldDir;
   struct DiskObject *Lock;
   struct WBArg *Arg;
   if (!(IconBase = (struct IconBase *)
         OpenLibrary("icon.library", 0L)))
      {
      printf("Library not received!\n");
      exit(FALSE);
      }
   for (i=0, Arg=WBenchMsg->sm_ArgList; i<WBenchMsg->sm_NumArgs;
        i++, Arg++)
      {
```

```
      printf("WBArg %d: Lock=0x%lx Name = %s\n",
             i, Arg->wa_Lock, Arg->wa_Name);
      if ((i == 0) && (Arg->wa_Lock ==0))
         {
         printf("Started without program. This was loaded
afterwards!\n");
         }
      else
         {
         OldDir = CurrentDir(Arg->wa_Lock);
         Lock = GetDiskObject(Arg->wa_Name);
         if (Lock != NULL)
            {
            FreeDiskObject(Lock);
            }
         CurrentDir(OldDir);
         }
      }
   printf("\nWAIT A MOMENT!\n");
   CloseLibrary(IconBase);
   Delay(5*60L);
   }
```

Program description

All this routine does is release the `DiskObject` and then return to the current directory. When we first access the text array contained in `ToolTypes`, we can see which values were assigned. Now let's replace the display routine:

```
ToolArray = Lock->do_ToolTypes;
j = 0;
do
   {
   printf("%d. Entry: %s\n", j, ToolArray[j]);
   j++;
   }
while(ToolArray[j] != Null);
```

Create multiple `ToolTypes` in an .info file for the next test. After saving, compiling and linking, return to the Workbench. Click on one of the Notepad texts and select the `Info` item from the `Workbench` menu. Click on the `ToolTypes` string gadget. `ToolTypes` are entered in the following manner:

```
TYPE = FLAGS
```

The word TYPE represents a keyword, which certain functions can access later on. You saw another example above in the form of a keyword named WINDOW:

```
WINDOW = CON:0/0/640/80/TestWindow
```

Keywords can consist of any alphanumeric characters.

Flags

Flags supply specific information about execution and other processes. The Notepad uses some flags that indicate whether the text uses one or more fonts (more on this later). When you wish to set more flags, separate each flag using the <|> character. This character is alternately known as the Or character.

So far we can read data using the above code. How can we display and edit this data? First we must see what the current values are in a program's .info file. Next, we must check for an additional icon that represents a Tool (application) or Project (data file). Project data takes precedence over Tool data.

The procedure is as follows. First we read the `ToolType`. For example, if we have FILETYPE present, it searches for the known types that our program also processes. We also compare to see if it is handled as an ASCII file that can be loaded from the corresponding routine. If it is not an ASCII file, the program must determine whether it can process the other format.

The following program examines the FILETYPE and displays a corresponding message. That is why you must prepare an .info file for this program, Use the notepad icon you created in the previous examples, remember to chage the default tool in the info window. This .info file has a WINDOW entry. Instead of the text output you can insert your own program name in the appropriate subroutines, or supply just the flags with values. This is especially advisable if more arguments are expected than just the file types.

```
/****************************************
 *                                      *
 * Program: Evaluate ToolTypes          *
 * ==================================== *
 * EvalToolTypes.c                      *
 * Author:  Date:       Comments:       *
 * ------  ----------  ----------       *
 * Wgb     06/20/1988  tests FILETYPE   *
 *                                      *
 *                                      *
 ****************************************/
#include <exec/types.h>
#include <workbench/workbench.h>
#include <workbench/startup.h>
#include <workbench/icon.h>
#include <stdio.h>
extern struct WBStartup *WBenchMsg;
extern struct IconBase *IconBase;
void                   *OpenLibrary();
main()
   {
   int i, j, Test;
   char **ToolArray, *Value;
   LONG OldDir;
   struct DiskObject *Lock;
```

```
    struct WBArg *Arg;
    if (!(IconBase = (struct IconBase *)
          OpenLibrary("icon.library", 0L)))
       {
       printf("Library not received!\n");
       exit(FALSE);
       }
    for (i=0, Arg=WBenchMsg->sm_ArgList; i<WBenchMsg->sm_NumArgs;
         i++, Arg++)
       {
       printf("WBArg %d: Lock=0x%lx Name = %s\n",
               i, Arg->wa_Lock, Arg->wa_Name);
       if ((i == 0) && (Arg->wa_Lock ==0))
          {
          printf("Started without program. This was loaded
afterwards!\n");
          }
       else
          {
          OldDir = CurrentDir(Arg->wa_Lock);
          Lock = GetDiskObject(Arg->wa_Name);
          if (Lock != NULL)
             {
             ToolArray = Lock->do_ToolTypes;
             Value = FindToolType(ToolArray, (char *)"FILETYPE");
             if (Value)
                {
                printf("ToolType FILETYPE with %s present!\n",
Value);
                Test = MatchToolValue(Value, (char *)"TOOLTEST");
                printf("Test result %d\n", Test);
                }
             else
                {
                printf("ToolType FILETYPE is not present!\n");
                }
             FreeDiskObject(Lock);
             }
          CurrentDir(OldDir);
          }
       }
    printf("\nWAIT A MOMENT!\n");
    CloseLibrary(IconBase);
    Delay(5*60L);
    }
```

The above program doesn't fulfill the set provisions, since it is only a short demo. Adding these features to your own programs would make using the Amiga much simpler for the user. The last example program makes the connection between CLI checks and .info file evaluations. It allows the user the possibility to set three flags: a(dd), p(rint) and i(nsert). This can occur through the CLI with a preceding hyphen (-a, -p, -io), or through the .info file (entering ADD, PRINT and INSERT next to the FLAGS keyword). Here's the listing:

```
/****************************************
 *                                      *
 * Program: Setting flags from CLI & WB *
 * ==================================== *
 *  FLAGS.C                             *
 * Author:  Date:       Comments:       *
 * ------  ----------  ----------       *
 * Wgb     06/20/1988  -a -p -i         *
 *                     FLAGS=           *
 *                     ADD|PRINT|INSERT *
 *                                      *
 ****************************************/
#include <exec/types.h>
#include <workbench/workbench.h>
#include <workbench/startup.h>
#include <workbench/icon.h>
#include <stdio.h>
extern struct WBStartup *WBenchMsg;
extern struct IconBase *IconBase;
void                   *OpenLibrary();
main(ArgC, ArgV)
int ArgC;
UBYTE *ArgV[];
   {
   int i, j;
   int TestA = 0, TestP = 0, TestI = 0;
   char **ToolArray, *Value;
   LONG OldDir;
   struct DiskObject *Lock;
   struct WBArg *Arg;
   /*********************************
    *                               *
    * Routine: CLI Reader           *
    * ============================= *
    *                               *
    * Author: Date:      Comments:  *
    * ------  ---------- ---------- *
    * Wgb     06/20/1988 -a -p -i   *
    *                               *
    *********************************/
   if (ArgC > 0)
      {
      for (i=0; i<ArgC; i++)
         {
         if (*ArgV[i] == (UBYTE)'-')
            {
            if (*(ArgV[i]+1) == 'a') TestA = TRUE;
            if (*(ArgV[i]+1) == 'p') TestP = TRUE;
            if (*(ArgV[i]+1) == 'i') TestI = TRUE;
            }
         }
      }
   else
   /****************************************
    *                                      *
    * Program section: WB Reader           *
    * ==================================== *
    *                                      *
    * Author:  Date:      Comments:        *
    * ------  ----------  ---------------- *
    * Wgb     06/20/1988  ADD|PRINT|INSERT *
    ****************************************/
```

```
        {
        if (!(IconBase = (struct IconBase *)
              OpenLibrary("icon.library", 0L)))
           {
           printf("Library not received!\n");
           exit(FALSE);
           }
        for (i=0, Arg=WBenchMsg->sm_ArgList; i<WBenchMsg-
>sm_NumArgs;
             i++, Arg++)
           {
           printf("WBArg %d: Lock=0x%lx Name = %s\n",
                   i, Arg->wa_Lock, Arg->wa_Name);
           if ((i == 0) && (Arg->wa_Lock ==0))
              {
              printf("Started without program. This was loaded
later!\n");
              }
           else
              {
              OldDir = CurrentDir(Arg->wa_Lock);
              Lock = GetDiskObject(Arg->wa_Name);
              if (Lock != NULL)
                 {
                 ToolArray = Lock->do_ToolTypes;
                 Value = FindToolType(ToolArray, "FLAGS");
                 if (Value)
                    {
                    TestA = MatchToolValue(Value, "ADD");
                    TestP = MatchToolValue(Value, "PRINT");
                    TestI = MatchToolValue(Value, "INSERT");
                    }
                 else
                    {
                    printf("ToolType FLAGS is not present!\n");
                    }
                 FreeDiskObject(Lock);
                 }
              CurrentDir(OldDir);
              }
           }
        printf("\nWAIT A MOMENT!\n");
        CloseLibrary(IconBase);
        /* Delay(5*60L); */
        }
     if (TestA) printf("ADD flag set!\n");
     if (TestP) printf("PRINT flag set!\n");
     if (TestI) printf("INSERT flag set!\n");
     Delay(5*60L);
     }
```

Program description

The program is composed of two main sections. The first section checks to see if arguments were entered from the CLI. If this is the case, these arguments are tested for the three relevant types. The flags are set correspondingly. These flags control output once the program ends.

Two factors to remember: First, the program doesn't test for unsupported flags. It simply displays an error message. In addition, we chose not to include an argument template as described earlier in this section. You can probably add these features later.

You can also assign values to a flag as well as assigning a flag itself. The format for this looks like the following:

```
Flags -w=20
```

This check also represents no problem if you use the `atoi()` function, which transforms the ASCII value "20" into the integer value 20. The second large section of the program executes when the program was started from the Workbench instead of the CLI. This routine examines `ToolTypes` and sets the corresponding flags. This check is incomplete, so it searches through only those `ToolTypes` affecting our flags. Other arrangements can be added here.

Here are some tables and rules for the .info files to conclude this chapter.

Sequence of Workbench messages:

1. Program info
2. First data file clicked on
3. Second data file clicked on

There is no info for a program if this is not clicked! That can be found out if the lock = 0.

Sequence of ToolTypes:

1. The `ToolTypes` are supplied in the abovementioned succession as blocks.
2. Individual blocks are transmitted in the same order as the entries in INFO.
3. `DefaultTool` of the first file clicked on is always loaded after the file; all others are ignored.

ToolTypes entries with the Notepad:

Name	Example	Meaning
FILETYPE	notepad	Notepad text
FONT	topaz.8	Global font
WINDOW	0,0,50,50	Window coordinates
FLAGS	NOGLOBAL	Set flags

Notepad flags:

Name	Meaning
NOGLOBAL	Disables global font function
GLOBAL	Enables global font function
NOWRAP	Disables word wrap
WRAP	Enables word wrap
NOFONTS	No font table loaded
FORMFEED	Enables form feed
DRAFT	Enables normal printing

4. Devices

You've probably heard the word device used by other Amiga programmers. Some of the source codes in this book access devices. This chapter will help you understand the purpose and practical application of devices.

4.1 What are devices?

Devices are actually nothing more than libraries containing additional information that pertains to device drivers. These drivers can control internal devices (e.g., disk drives, keyboard and game ports) and external devices (e.g., printers and modems). Devices can be opened like libraries. Here are the `Exec` functions used to call libraries and devices:

Libraries:	`OpenLibrary()`
Devices:	`OpenDevice()`

Libraries vs. devices

There is another major difference between devices and libraries. When opening a library, `OpenLibrary()` returns the address of a completely initialized and ready-to-enter library. `OpenDevice()` also reports eventual errors to the programmer. That is why `OpenDevice()` needs pre-initialized structures from the user. The structures can be joined together and completely initialized.

Let's look at a typical `OpenDevice()` call:

```
Error = OpenDevice("trackdisk.device", 0L, DiskRequest, 0L);
if (Error != 0) CloseIt   ("OpenDevice() - Error");
```

The structure controlling the device (in this case, the internal disk drive) is an `IORequest` structure:

```
Offsets   struct IORequest
-------   { /* defined in "exec/io.h" */
0  0x00      struct Message  io_Message;
20  0x14     struct Device  *io_Device;
24  0x18     struct Unit    *io_Unit;
28  0x1c     UWORD           io_Command;
30  0x1e      UBYTE           io_Flags;
31  0x1f      BYTE            io_Error;
32  0x20   } /* 32 == NumBytes of this structure */
```

IORequest

You don't need to define the IORequest structure in your program every time you want to use it. This structure is defined in the include file "exec/io.h". Once called, this structure lies ready for your declarations. For example:

```
struct IORequest *DiskRequest;
```

Device initialization

We already mentioned that the device structures must be initialized before use, unlike the libraries. This is done using the Exec support function CreatExtIO(). "Exec support function" means that this function is not based in a system library. Instead, it can be found in the linker library for your compiler (c.lib for Aztec C or amiga.lib for Lattice C). Here's a simple example of how CreatExtIO() works:

```
/*******************************************************/
/*                  CreateExtIO()          (Exec support)*/
/*                                                     */
/* Function:    Create device block                    */
/*-----------------------------------------------------*/
/* IOReplyPort: MsgPort for WaitIO() etc.              */
/* Size:        Size of the device block               */
/*******************************************************/
 struct IORequest *CreatextIO(IOReplyPort, Size)
struct MsgPort             *IOReplyPort;
LONG                                      Size;
{
     struct IORequest *Request;
     /* no IOReplyPort? then leave CreateExtIO() */
     if (IOReplyPort == NULL) return(NULL);
     /* reserves memory for request */
     Request = AllocMem(Size, MEMF_PUBLIC | MEMF_CLEAR);
     /* no memory? Then leave CreateExtIO() */
     if (Request == NULL) return(NULL);
     /* report of type MESSAGE */
     Request->io_Message.mn_Node.ln_Type = NT_MESSAGE;
     /* reports are SIZE bytes long */
     /* See also DeleteExtIO().        */
     Request->io_Message.mn_Length       = Size;
     /* give port messages */
     Request->io_Message.mn_ReplyPort = IOReplyPort;
     return(Request);
}
```

CreateExtIO() allocates memory for a device request structure. Standard device blocks such as IOStdReq, IORequest, IOAudio and others fall under the category of device request structures. You can store and initialize these different device blocks, which help the communication between computer and device, using CreatExtIO().

Common ground

All of the device request structures or device blocks have one thing in common: The first element is an IORequest structure. This means that all of the different device blocks can be handled the same from CreateExtIO(). The difference first appears after the first few bytes of the IORequest structure. Let's look at the IOAudio structure (the audio device block):

```
struct IOAudio
{
   struct IORequest  ioa_Request;/* IORequest at the beginning*/
   WORD              ioa_AllocKey;
   UBYTE            *ioa_Data;
   ULONG             ioa_Length;
   UWORD             ioa_Period;
   UWORD             ioa_Volume;
   UWORD             ioa_Cycles;
   struct Message    ioa_WriteMessage;
}
```

The first element of the IOAudio structure is an IORequest structure. When you assign CR=CreateExtIO() the starting address of some IOAudio structure, this address is identical to that of an IORequest structure. Because all structure element accesses occur through offsets, they are automatically initialized at the beginning of the IORequest structure of the IOAudio block. We can initialize it using the following I/O block to open the device:

```
struct IOAudio *OwnIOAudio;
...
OwnIOAudio = (struct IOAudio *)
              CreateExtIO(AudioPort, sizeof(struct IOAudio))
if (OwnIOAudio == 0) CloseIt("CreateExtIO() - Error");
...
```

The block is then executed with the help from OpenDevice():

```
OpenDevice("audio, device", 0L, OwnIOAudio, 0L));
```

This sample call shows the purpose of the sizeof parameter for CreateExtIO(). It gives the size of the I/O blocks to be allocated.

IOReplyPort

The first parameter for CreateExtIO() (IOReplyPort) requires closer examination. IOReplyPort is nothing more than a message port. Message ports are needed for message reports. They act as anchor points for the devices. They hold reports to the system just as an anchor holds a ship to the bottom of the sea. These ports must be initialized before they can be used:

```
struct MsgPort *AnyOldPort;
AnyOldPort = (struct MsgPort *)
     CreatePort("anyold.port", 0);
if (anyoldport == 0) CloseIt("CreatePort() - Error");
...
```

The routine used for this is called `CreatePort()`. It is also an Exec support function:

```
/*********************************************************/
/*                  CreatePort()          (Exec support)*/
/*                                                      */
/* Function: Create MessagePort                         */
/*------------------------------------------------------*/
/* Name:      Name of the MsgPort                       */
/* Priority: Priority of the port                       */
/*********************************************************/
struct MsgPort *CreatePort(Name, Priority)
char                      *Name;
BYTE                              Priority; {
     struct MsgPort *Port;
     BYTE           SignalBit;
     if ((SignalBit = AllocSignal(-1)) == -1) return(NULL);
     /* no signal bit can be reserved */
     Port = (struct MsgPort *)
       AllocMem(sizeof(struct MsgPort), MEMF_PUBLIC|MEMF_CLEAR);
     /* memory allocation */
     if (Port == NULL)
     {
          Freesignal(SignalBit);
          return(NULL);
     }
     /* no memory */
     Port->mp_Node.ln_Name = Name;
     Port->mp_Node.ln_Pri  = Priority;
     Port->mp_Node.ln_Type = NT_MSGPORT;
     Port->mp_Flags        = PA_SIGNAL;
     Port->mp_SigBit       = SignalBit;
     Port->mp_SigTask      = FindTask(0l);
     /* Initialization */
     if (Name != 0L) AddPort(Port);
     else            NewList(&(Port->mp_MsgList));
     /* Port in new list */
     return(Port);
}
```

When using the devices the message ports generally have no name, so the initialization of a message port can be done by using the following sequence:

```
struct MsgPort *Port;
Port = (struct MsgPort *)CreatePort(0L, 0L);
```

`CreatePort()` throws out the "anchor" on which the device secures itself through `CreateExtIO()` and `OpenDevice()`:

```
struct MsgPort   *Port;
struct IORequest *Request;
Port    = (struct MsgPort *)   CreatePort(0L, 0L);
Request = (struct IORequest *) CreateExtIO(Port, sizeof(struct
          IORequest));
OpenDevice;(DEVICENAME, 0L, Request, 0
...
```

The three routines listed above give you the power to access any device. Because the steps for opening a device are always the same, we'll now list some universal routines for allocating and de-allocating device blocks, for opening and closing devices:

```
/**********************************************************/
/*                 Device-Support Functions               */
/*                  (c) Bruno Jennrich                    */
/*                                                        */
/*                      June 8 1988                       */
/**********************************************************/
/**********************************************************/
/* Compile Info:                                          */
/*--------------------------------------------------------*/
/*                                                        */
/* cc Devs_Support                                        */
/**********************************************************/
#include "exec/types.h"
#include "exec/io.h"
#include "exec/devices.h"
VOID  CloseIt();                  /* CloseIt() exists    */
                                  /* in your own program */
VOID *CreatePort();               /* Exec-Support        */
VOID *CreateExtIO();
VOID  DeletePort();
VOID  DeleteExtIO();
/**********************************************************/
/*                  GetDeviceBlock()                      */
/*                                                        */
/* Function:    Device-Block open and initialization      */
/*--------------------------------------------------------*/
/* Input - Parameter:                                     */
/*                                                        */
/* Size:        Size of the Device-Blocks in bytes        */
/*--------------------------------------------------------*/
/* Return value:                                          */
/*                                                        */
/*              Initialize Device-Block                   */
/**********************************************************/
APTR GetDeviceBlock (Size)
ULONG                Size;
{
   struct MsgPort *Device_Port;
   APTR           Device_Request;
   /* Becasue this routine should be insertable universally*/
   /* no IORequest-Structure is placed, but instead       */
   /* any structure that through  (CASTS) can be          */
   /* passed to  the IORequest-Structure.                 */
   /* Tries to allocate the Device-Port. If this is not   */
```

```
   /* possible , leave program. (CloseIt()).                  */

   Device_Port = (struct MsgPort *) CreatePort (0,0);
   if (Device_Port == 0) CloseIt ("Couldn4t get DEVICE-PORT !");
   /* Tries to allocate Device-Block. If that is not          */
   /* possible, give Device-Port back and leave               */
   /* program.                                                */

   Device_Request = (APTR) CreateExtIO (Device_Port, Size);
   if (Device_Request == 0)
   {
      DeletePort (Device_Port);
      CloseIt ("Couldn4t get DEVICE-BLOCK !");
   }

   /* Give back previously installed Device-Block             */
   return (Device_Request);
}
/************************************************************/
/*                    FreeDeviceBlock()                      */
/*                                                          */
/* Function:    Release Device-Block                        */
/*----------------------------------------------------------*/
/* Input - Parameter:                                       */
/*                                                          */
/* IORequest:   Release Device-Block                        */
/************************************************************/
VOID FreeDeviceBlock (IORequest)
struct IORequest     *IORequest;
{
   /* If IORequest can be opened, free up                   */
   /* Device-Port. The free up IORequest                    */

   if (IORequest != 0)
   {
      if (IORequest->io_Message.mn_ReplyPort != 0)
         DeletePort  (IORequest->io_Message.mn_ReplyPort);
      DeleteExtIO (IORequest);
   }
}
/************************************************************/
/*                    Open_A_Device()                        */
/*                                                          */
/* Function:    Open any Device                             */
/*----------------------------------------------------------*/
/* Input - Parameter:                                       */
/* Name:           Name the Devices (i.e. "audio.device")   */
/* Unit:           Device-Unit                              */
/* Device_Request: Pointer to block to be initialized       */
/*                 (initialized) Device-Block               */
/* Flags:          Device-Flags                             */
/* Size:           Size of the Device-Blocks                */
/************************************************************/
VOID Open_A_Device (Name, Unit, Device_Request, Flags, Size)
char               *Name;
ULONG                     Unit;
APTR                            *Device_Request;
ULONG                                          Flags, Size;
{
   UWORD Error;     /* Error from OpenDevice()  */
   /* If Size > 0, allocate Device-Block.        */
   /* If Size == 0, use initialized device block*/
```

```
   /* from user                              */
   if (Size != 0) *Device_Request = GetDeviceBlock(Size);

   /* Open Device */
   Error = OpenDevice (Name, Unit, *Device_Request, Flags);
   if (Error != 0)
   {
      printf ("Open-Device Error #%41x\n",Error);
      CloseIt ("Couldn4t get DEVICE !");
   }

   /* NOTE !!!      4Device_Request4 is a pointer   */
   /*                to a pointer ! (**DevReq)      */
}
/************************************************************/
/*                    Close_A_Device                        */
/*                                                          */
/* Function:    Device-Block free and close Device          */
/*----------------------------------------------------------*/
/* Input - Parameter:                                       */
/*                                                          */
/* IORequest:   Released Device-Block                       */
/************************************************************/
VOID Close_A_Device (IORequest)
struct IORequest    *IORequest;
{
   /* If IORequest can be opened, release      */
   /* Device-Port. Close Device. The release   */
   /* IORequest.                               */

   if (IORequest != 0)
   {
      if (IORequest->io_Message.mn_ReplyPort != 0)
         DeletePort  (IORequest->io_Message.mn_ReplyPort);
      if (IORequest->io_Device != 0)
         CloseDevice (IORequest);
      DeleteExtIO  (IORequest);
   }
}
/************************************************************/
/*                      Do_Command()                        */
/*                                                          */
/* Function:    Execute command                             */
/*----------------------------------------------------------*/
/* Input - Parameter:                                       */
/*                                                          */
/* DeviceBlock:  Device-Block                               */
/* Command:      command                                    */
/************************************************************/
VOID Do_Command  (DeviceBlock,Command)
struct IORequest *DeviceBlock;
UWORD                        Command;
{
   DeviceBlock->io_Command = Command;
   DoIO(DeviceBlock);
}
```

Closing a device

You can open devices and then close them again with the help of these functions. You must close an open device when finished with it, just as you must close a library. While programs and libraries may be used at the same time, this is not possible with devices. The user can only access one device at a time. For other users to obtain access, the devices must be closed after they have been used. This happens through the `Exec` support routines `DeleteExtIO()` and `DeletePort()`, as well as the `CloseDevice()` command:

```
/***********************************************************/
/*                   DeletePort()           (Exec support)*/
/* Function:     Release port                              */
/*---------------------------------------------------------*/
/* Input parameters:                                       */
/* IOReplyPort:  Released port                             */
/***********************************************************/
VOID DeletePort(IOReplyPort)
struct MsgPort *IOReplyPort;
{
   /* if port is nameless, then remove port */
   if ((IOReplyPort->mp_Node.ln_Name) != 0L)
      RemPort(IOReplyPort);
   /* erase type of port */
   IOReplyPort->mp_Node.ln_Type = 0xff;
   /* remove port from list */
   IOReplyPort->mp_MsgList.lh_Head = (struct Node *)-1;
        /* Release memory of port */
   FreeMem(IOReplyPort, sizeof(struct MsgPort));
}
```

`DeleteExtIO()` looks like the following:

```
/*********************************************************/
/*                  DeleteExtIO()         (Exec support)*/
/* Function:  Release device block                      */
/*-------------------------------------------------------*/
/* Input parameters:                                     */
/* IORequest: Device block to be released                */
/*********************************************************/
DeleteExtIO (IORequest)
struct IORequest *IORequest;
{
   /* In case IORequest is not present, leave routine */
   /* otherwise freed twice alert arises     */

   if (IORequest == 0) return(0l);
     /* IORequest mutilated so that further */
   /* use is impossible                               */
   IORequest->io_Message.mn_Node.ln_Type = 0xff;
   IORequest->io_Device = (struct Device *)-1;
   IORequest->io_Unit   = (struct Unit *)-1;
   /* memory freed up. (memory from FreeMem() */
   /* not erased, that is why the above mutilation.)    */
   FreeMem(IORequest, IORequest->io_Message.mn_Length);
}
```

In addition to these two `Exec` support functions, which make it impossible to re-use `IOReplyPorts` and `IORequest` structures, we have used the command `CloseDevice()` in our device support functions. This ensures that the device can be used from other programs again.

The device to be closed is informed through `CloseDevice()` over the `IORequest` block, which contains a pointer to the opened device (`IORequest->io_Device`). This pointer extracts `CloseDevice()` itself and uses it for the closing. When you want to use the routines from `Devs_Support`, you make a routine with the name `CloseIt()` available for use. This is always called when an error is encountered while opening a device. You will encounter such `CloseIt()` routines in this book, and we want you to have confidence in the demands that this routine makes:

```
/*****************************************************/
/*                   CloseIt()                (User) */
/* Function:  display encountered error.             */
/*            Close everything.                      */
/*---------------------------------------------------*/
/* Input Parameters:                                 */
/* String:     Error String                          */
/*****************************************************/
VOID CloseIt(String)
char        *String;
{
     UWORD Error   = 0;
     UWORD i;
     UWORD *dff180 = (UWORD *)0xdff180;

     if (strlen(String) > 0)
     {
        for(i=0;i<0xffff;i++) *dff180 = i;

        puts(String);
        puts("\n");
        Error = 100;
     }
     /* free-up routine */
     exit(Error);  /* leave program */
}
```

This `CloseIt()` routine ensures that the screen blinks colorfully once by specifying the background color registers. Then the error message appears on the screen, telling you the location at which you should look for an error. Then all of the opened device libraries, etc., are released. You should make sure that everything is released before the error message is displayed. Most errors occur when closing, so you don't know the error that most recently occurred.

Warning:

You should only use `CloseIt()` as an "emergency exit." You should never use `CloseIt()` for convenience as the exit of the program, which frees all of the structures for you. Rule of thumb: The routine that allocates memory also releases memory.

4.2 Communicating through devices

You now know how to open a device and how to close it again. Now let's look at how you can exchange data between a program and a device.

A device has many similarities to a library. When the device is opened, `IORequest->io_Device` provides some routines for establishing communication between program and device. This device library contains the following commands and other data:

```
Command:    Offset:
-------     -------
Open        -0x06
Close       -0x0c
Expunge     -0x12
Extfunc     -0x18
BeginIO     -0x1e
AbortIO     -0x24
```

Open

`Open` is a routine called from `OpenDevice()` to control the device specific installations. The `Exec` function `OpenDevice()` offers you access to this device file. The device's own `Open` command provides the necessary steps for the initialization of the addressed device.

Close

`Close` is called from `CloseDevice()` to make sure that the device file closes properly.

Expunge

If a device must be loaded from disk, the memory allocated for the device structures is released by `Expunge`. `Open` and `Close` increment and decrement the allocated amount. The occupied memory is not completely released by `Close`. `Expunge` releases all memory once it is called from the `Exec` function `RemDevice()`.

Extfunc

The `Extfunc` routine is reserved for special tasks (e.g., printer device DO_SPECIAL). The routines `BeginIO` and `AbortIO` are the routines we are interested in:

BeginIO

`BeginIO` initiates the data transfer between the program and the device. After the device block is completely initialized (data pointer adjusted, command given, etc.), this command is called from `SendIO()` as well as from `DoIO()`. The call looks like this:

```
BeginIO:
                        * A1 contains *IORequest  *
move.l   20(a1), a6   * Device Library after  A6 *
jmp         -$1e(a6) * jump to BeginIO      *
```

The actual `BeginIO` routine, which is jumped to here through your offset, looks for every other device. This is also logical, because different steps, other than the execution when using the trackdisk.device, are taken for the use of timer devices. `BeginIO` is also called from `SendIO()` and `DoIO()` (the command `BeginIO()` is also in the linker library for your compiler [`c.lib` for Aztec C or `amiga.lib` for Lattice C]). What difference is there between `DoIO()` and `SendIO()`?

SendIO

`SendIO()` is an asynchronous command. That means that the called program can continue its processing after a device command is sent. The device command is executed alongside the called program. When using `DoIO()`, the program that was called must wait until the command from the device is completely processed. That is why you name the command `DoIO()`. Program execution and the device command are synchronized.

`DoIO()` and `SendIO()` after identical in their handling of a device command. The ending of the device command is expected by means of `WaitIO()` when using `DoIO()` in conjunction with the `MsgPorts`. After construction you can use `DoIO()` by means of `SendIO()` and `WaitIO()`:

```
Second_DoIO       (IORequest)
struct IORequest *IORequest;
{
   SendIO(IORequest);
   WaitIO(IORequest);
}
```

`WaitIO()` can be constructed by means of `CheckIO()`:

```
Second_WaitIO     (IORequest)
struct IORequest *IORequest;
{
   while (CheckIO (IORequest)==0);
   /* Device command is not ended */
}
```

When the device command is fully operational, `CheckIO()` returns the value 0 in register D0. When the command is processed, the address of the device block (`IORequest`) after `CheckIO()` passes to register D0. You now know the functions needed for communication between a device and the program.

Now for a question: If you would like to send a command to the device as things currently stand, which variable of the `IORequest` (or

IOStdRequest) structure comes into play? We'll discuss details as we examine each device. For now, here are a few items common to each device:

- IORequest->io_Command is the variable into which the device command is stored.
- The device command consists of only one number. This number branches in SendIO() or DoIO() for some device specific routines.

Just stating the command does nothing. When you want to include data, a data pointer and a variable supply the length or number of data bytes to be transferred. The transfer of data is no longer possible with the help of the simple IORequest structure. The IOStdReq structure or a device specific structure (see IOAudio) must be used instead:

```
Offsets  struct  IOStdReq
-------    {/* defined in "exec/io.h" */
              /* - IORequest - */
 0  0x00      struct Message io_Message;
20  0x14      struct Device  *io_Device;
24  0x18      struct Unit    *io_Unit;
28  0x1c      UWORD  io_Command;
30  0x1e      UBYTE  io_Flags;
31  0x1f      BYTE   io_Error;
              /* - IOStdReq - */
32  0x20      ULONG  io_Actual;
36  0x24      ULONG  io_Length;
40  0x28      APTR   io_Data;
44  0x2c      ULONG  io_Offset;
48  0x30    }
```

The data pointer which works in conjunction with a structure has a similar name. For example, the data pointer of the IOStdReq structure is called IOStdReq.io_Data, while the data pointer of the IOAudio structure is called IOAudio.ioa_Data. The number of data bytes for IOStdReq can be found in IOStdReq.io_Length. IOStdReq.io_Actual often specifies the number of data bytes written or read. These variables occur only with more complex device blocks such as the IORequest block.

Flag variables

Flag and error variables also appear in device blocks. You can control the execution of a device command with the help of flag variables. Because almost every device has its own flags, these flags can specify when it is necessary to use the device. All devices have the IOF_QUICK flag in common. This IOF_QUICK flag is set if a command can be processed immediately. The read and write commands of the trackdisk.device, for example, go in a trackdisk task. The program that was called must wait for this command to be processed with assistance from a message port. When reading disk status (e.g., write protect on), a set IOF_QUICK flag may not occur, since this command is executed without the trackdisk task.

Error variables

The error variable is of particular importance. If the error variable equals zero after a device command, everything is running all right. When the error variable contains a value other than zero, an error occurred. You should react according to the severity of the error (e.g., a warning to the user). Each device has its own errors.

The following errors are common to all devices:

Variable	Value	Meaning
IOERR_OPENFAIL	(-1)	Device cannot be opened
IOERR_ABORTED	(-2)	Command interrupted by AbortIO()
IOERR_NOCMD	(-3)	invalid command
IOERR_BADLENGTH	(-4)	io_Length has an invalid value

Almost every device uses the read (CMD_READ) and write (CMD_WRITE) commands. The reset command (CMD_RESET), which places the device in the original condition, is also used by almost every device.

Extended commands

In addition to these standardized commands, each device has its own extended commands. These commands capitalize on the individual device's special abilities. The remaining sections of this chapter describe these commands and how they affect their devices. In the following sections we listed the easiest device access routines we know. You'll find device specific support routines as well as generic device support routines. These routines need a device block and parameters for you to access them. This saves you the trouble of assigning values to structure elements.

4.3 The parallel device

The parallel device allows easy access to the Amiga's parallel interface. You can read and write data through the parallel interface.

The parallel device supports the following commands:

CMD_RESET	(1)	resets device to post-OpenDevice() status (including TermArray)
CMD_READ	(2)	reads data
CMD_WRITE	(3)	writes data
CMD_STOP	(6)	stops read/write (expects handshake)
CMD_START	(7)	restarts read/write
CMD_FLUSH	(8)	ignores existing read/write commands

The parallel device includes two device specific commands:

PDCMD_SETPARAMS	(9)	sets parameters
PDCMD_QUERY	(10)	finds out port status

4.3.1 Opening the parallel device

The following code easily opens the parallel device:

```
struct IOExtPar *ParReq = 0L;
#define PAR_LEN (ULONG) sizeof(struct IOExtPar)
...
  Open_A_Device("parallel.device", 0L, ParReq, 0L, PAR_LEN);
...
```

If another user opens the parallel device, you no longer have access. Setting the PARB_SHARED flag (32) before you open the device ensures that multiple users can access the parallel device at one time:

```
struct IOExtPar *ParReq = 0L;
#define PAR_LEN (ULONG) sizeof(struct IOExtPar)
VOID *GetDeviceBlock();
...
  ParReq = (struct IOExtPar*)GetDeviceBlock(PAR_LEN);
  ParReq->io_ParFlags = (UBYTE) PARB_SHARED;
  Open_A_Device("parallel.device", 0L, ParReq, 0L, 0L);
...
```

Problems with sharing

Transferring data from multiple programs to the parallel device can cause problems. For example, texts sent from two different word processors may run together through the device.

Now back to the `Open_A_Device()` command. You must assign this address to a pointer to the device block (&`ParReq`—`ParReq` is its pointer). The parallel device's device block looks like the following:

```
Offset    Structure
------    --------
          struct IoExtPar
          {
0  0x00     struct IOStdReq IOPar;
48 0x30     ULONG           io_PExtFlags; /* unused */
52 0x32     UBYTE           io_Status;    /* Port Status */
53 0x33     UBYTE           io_ParFlags;  /* SHARED+EOFMODE */
54 0x34     struct IOPArray io_PTermArray;/* Terminates */
62 0x3c   } /* defined in "devices/parallel.h" */
```

4.3.2 Writing parallel device data

All you need to write data to the parallel device is the data to be sent and the number of data bytes to be sent. Once those items are established, you can invoke `CMD_WRITE`:

```
/***************************************************************
*                    Parallel_Write()            (Par_Support)*
*                                                              *
* Function: Send data over the parallel interface              *
*--------------------------------------------------------------*
* Input - Parameter:                                           *
*                                                              *
* ParReq:   Device-Block                                       *
* Data:     Date to be sent                                    *
* Len:      number of bytes to be sent                         *
***************************************************************/

VOID Parallel_Write (ParReq,Data,Len)
struct IOExtPar     *ParReq;
APTR                       Data;
ULONG                           Len;
{
   ParReq->IOPar.io_Data    = Data;
   ParReq->IOPar.io_Length  = Len;
   Do_Command (ParReq, (UWORD) CMD_WRITE);
}
```

If you give the value -1 for Len (the number of bytes to be written), the parallel device writes data until it encounters a null byte. The parallel device writes this null byte and stops writing data.

4.3.3 Reading parallel device data

The following routine shows how to read data from the parallel device:

```
/***************************************************************
*                      Parallel_Read()           (Par_Support)*
*                                                              *
* Function: Read data from the parallel interface              *
*--------------------------------------------------------------*
* Input - Parameter:                                           *
*                                                              *
* ParReq:   Device-Block                                       *
* Data:     Data buffer                                        *
* Len:      Number of bytes to be read                         *
***************************************************************/

VOID Parallel_Read (ParReq,Data,Len)
struct IOExtPar    *ParReq;
APTR                     Data;
ULONG                         Len;
{
   ParReq->IOPar.io_Data    = Data;
   ParReq->IOPar.io_Length  = Len;
   Do_Command (ParReq, (UWORD) CMD_READ);
}
```

You can stop the character reading process using a defined character. The `IOPArray` can contain the eight terminators:

```
Offset  Structure
------  -------
        struct IOPArray
        {
0  0x00    ULONG PTermArray0;
4  0x04    ULONG PTermArray1;
8  0x08 }; /* defined in "devices/parallel.h" */
```

The eight terminators are specified as two long words:

```
Parallel_SetParams(ParReq, PARB_EOFMODE, 0x00010101,
0x01010101);
/* PARB_EOFMODE = 2 */
```

Terminators

When the terminators of the `Parallel_SetParams()` function are determined by the method listed above, the reading stops after the routine encounters 0x00 or 0x01 in ASCII code form. When you, as above, establish less than eight terminators, you should pad the remaining terminators with the value of the last terminator listed.

The `Parallel_SetParams()` function looks like the following:

```
/***************************************************************
*                  Parallel_SetParams()         (Par_Support)*
*                                                             *
* Function: Change interface parameters                       *
*-------------------------------------------------------------*
* Input - Parameter:                                          *
* ParReq:        Device-Block                                 *
* Flags:         new Flags                                    *
* TermArray0/1: new terminators                               *
***************************************************************/

VOID Paralell_SetParams (ParReq,Flags,TermArray0,TermArray1)
struct IOExtPar        *ParReq;
BYTE                           Flags;
ULONG                                  TermArray0;
ULONG                                               TermArray1;
{
   ParReq->io_ParFlags = Flags;
   ParReq->io_PTermArray.PTermArray0 = TermArray0;
   ParReq->io_PTermArray.PTermArray1 = TermArray1;

   Do_Command (ParReq,(UWORD)PDCMD_SETPARAMS);
}
```

After `Open_A_Device()` the `PTermArray` is ignored and only 0x00 is recognized as a terminator. Logically you can only change the parameter for the parallel device if there is no write or read access. To change the parameters of such an operation destroys the communication base between the sender and the receiver, making further communication impossible. For now, you can only change the terminators with `Parallel_SetParams()` (`PARB_EOFMODE` must be set accordingly).

4.3.4 Reading parallel device status

You can read the status of the interface in `io_Status`, with the help of the `PDCMP_QUERY` command. The bits in `io_Status` represent the following:

```
Bit 0       IOPTF_PSEL = 1 (Printer Selected)
            = 1: OFFLINE
            = 0: ONLINE
Bit 1       IOPTF_PAPEROUT = 2
            = 1: OK
            = 0: PAPER_OUT
Bit 2       IOPTF_PBUSY = 4 (Printer busy)
            = 1: Printer has nothing to do
            = 0: Printer printed
Bit 3       IOPTF_RWDIR = 8 (Direction (Read, Write))
            = 1: It was written
            = 0: It was read
Bit 4-7     reserved
```

Bits 0, 1, and 2 are active low. There is a 0 in the flag labeled Status. PaperOut indicates that the flag IOTPF_PAPEROUT is not set (these flags are defined in devices/parallel.h). You have already seen that the parallel device is often assigned to an interfaced and active printer. Basically, the above bits can be used for other devices such as an EPROM burner. This status byte informs the computer that the parallel device is not ready to begin processing the data it received.

The following command sequence displays the status of the parallel device:

```
Do_Command(ParReq, (UWORD)PDCMD_QUERY);
Status = ParReq->io_Status;
```

4.3.5 A parallel device application

The following program sends a short string through the parallel device. There is usually a printer connected to the parallel port. The program can tell whether a printer is actually connected to this port, or whether the printer is switched off. If the printer is off or not connected, the program displays the message "Printer OFFLINE or not ready" on the screen. Combine the three routines in the previous section to form the Par_Support.c module, don't forget to include the exec/types.h, exec/memory.h, exec/io.h, and devices/printer.h files in Par_Support.c.

```
/****************************************************************
*                          Par.c                          (User)*
*                  (c) Bruno Jennrich                          *
*                       August 1988                            *
****************************************************************/
/****************************************************************
* Compile-Info:                                                *
* cc Par.c                                                     *
* ln Par.o Par_Support.o Devs_Support.o -lc                    *
****************************************************************/
#include "exec/types.h"
#include "exec/memory.h"
#include "exec/io.h"
#include "devices/parallel.h"
struct IOExtPar *ParReq = 0l;
#define PAR_LEN (ULONG) sizeof (struct IOExtPar)
VOID *GetDeviceBlock();
/****************************************************************
*                        CloseIt()                        (User)*
*                                                              *
* Function: If error, close all                                *
*--------------------------------------------------------------*
* Input - Parameter:                                           *
* String: Error-Message                                        *
****************************************************************/
VOID CloseIt (String)
```

```
char         *String;
{
   UWORD i;
   UWORD *dff180 = (UWORD *)0xdff180;
   UWORD Error = 0;
   if (strlen (String) > 01)
   {
      for (i=0;i<0xffff;i++)  *dff180 = i;
      puts (String);
      Error = 10;
   }
   if (ParReq != 01) Close_A_Device (ParReq);
   exit (Error);
}
/***************************************************************
*                          main()                        (User)*
***************************************************************/
main ()
{
   BYTE *String = "I write this to the Parallel-Port\015";
   ParReq = (struct IOExtPar *)GetDeviceBlock (PAR_LEN);
   ParReq->io_ParFlags = (UBYTE) PARF_SHARED;
   Open_A_Device ("parallel.device",01,&ParReq,01,01);
   Do_Command (ParReq,(UWORD) PDCMD_QUERY);
   if (((UBYTE)ParReq->io_Status & (UBYTE) IOPTF_PSEL) ==
(UBYTE) IOPTF_PSEL)
      printf ("Printer OFFLINE or not ready !\n");
   else
      Parallel_Write (ParReq,String,(ULONG)strlen (String));
   Close_A_Device (ParReq);
}
```

The Selected pin (pin 13) of the Centronics port usually indicates whether or not a printer is connected. If this pin still equals zero after thirty seconds, the program aborts.

4.3.6 Parallel device error messages

The following errors can be encountered when using the parallel device:

`ParErr_DevBusy`	(1)	Parallel device busy. Non-functional `PDCMD_SETPARAMS`.
`ParErr_BufToBig`	(2)	Read/write buffer too large.
`ParErr_InvParam`	(3)	This parameter change not implemented in this version. Only `PARB_EOFMODE` currently allowed for terminator changes.
`ParErr_LineErr`	(4)	Transfer error.
`ParErr_NotOpen`	(5)	Error occurred when opening the device (e.g., parallel.device not in the devs drawer of the SYS disk). Error occurred during `OpenDevice()`.

`ParErr_PortReset`	(6)	`parall.interface` reset.
`ParErr_InitErr`	(7)	Error occurred during parallel device initialization (`OpenDevice()`).

4.3.7 Centronics port pin arrangement

Pin	A500	A1000	A2000	
1	STROBE	DRDY	STROBE	
2	Data0	Data0	Data0	
3	Data1	Data1	Data1	
4	Data2	Data2	Data2	
5	Data3	Data3	Data3	
6	Data4	Data4	Data4	
7	Data5	Data5	Data5	
8	Data6	Data6	Data6	
9	Data7	Data7	Data7	
10	ACK	ACK	ACK	
11	BUSY	BUSY	BUSY	
12	POUT	POUT	POUT	(Paper Out)
13	SEL	SEL	SEL	(Selected == OnLine)
14	+5v	GND	+5v	
15	NC	GND	NC	
16	RESET	GND	RESET	
17	GND	GND	GND	
18	GND	GND	GND	
19	GND	GND	GND	
20	GND	GND	GND	
21	GND	GND	GND	
22	GND	GND	GND	
23	GND	+5v	GND	
24	GND	NC	GND	
25	GND	RESET	GND	

The Amiga 500 and 2000 require a DB25 male plug for connection to their parallel ports.

Note: Never use a standard IBM printer cable alone on an Amiga 1000. The Amiga 1000 uses a reverse standard, and inserting such a cable may destroy your computer. Purchase a gender changer to reverse the Centronics pinout to normal (the Amiga 1000 parallel port accepts a DB25 female connector. Connect the gender changer to the Amiga 1000 parallel port, then connect a standard IBM printer cable to the exposed end of the gender changer.

4.4 The serial device

The serial device allows access to the serial interface of the Amiga. A few device blocks also exist here:

```
Offset    Structure
------    --------
          struct IOExtSer
          {
0  0x00      struct IOStdReq IOSer;
48 0x30      ULONG           io_CtlChar; /* transfer protocol */
52 0x34      ULONG           io_RBufLen; /* Read buffer size */
56 0x38      ULONG           io_ExtFlags;/* unused */
60 0x3c      ULONG           io_Baud;    /* Baud rate */
64 0x40      ULONG           io_BrkTime; /* Break time */
68 0x44      struct IOTArray io_TermArray;/* Terminators */
76 0x4c      UBYTE           io_ReadLen; /* 7 or 8 Bits */
77 0x4d      UBYTE           io_WriteLen;/* 7 or 8 Bits */
78 0x4e      BYTE            io_StopBits;/* 0, 1, 2 */
79 0x4f      BYTE            io_SerFlags;/* see SetParams */
80 0x50      UWORD           io_Status;  /* see Query */
82 0x52   } /* defined in "devices/serial.h" */
```

Notice the terminator array (see Section 4.1). This array also consists of eight bytes or two long words:

```
Offset    Structure
------    --------
          struct IOTArray
          {
0  0x00      ULONG TermArray0;
4  0x04      ULONG TermArray1;
8  0x08   } /* defined in "devices/serial.h" */
```

Now we come to the commands which the serial device understands:

MD_RESET	(1)	resets device to post-OpenDevice() status (including TermArray)
CMD_READ	(2)	reads data
CMD_WRITE	(3)	writes data
CMD_STOP	(6)	stops reading/writing
CMD_START	(7)	continue read/write operation
CMD_FLUSH	(8)	ignores existing read/write commands

In addition there are three device specific commands:

SDCMD_QUERY	(9)	finds out port status
SDCMD_BREAK	(10)	stops transfer
SDCMD_SETPARAMS	(11)	sets parameters

4.4.1 Opening the serial device

The following code easily opens the serial device:

```
struct IOExtSer *SerReq = 0L;
#define SER_LEN (ULONG) sizeof(struct IOExtSer)
...
   Open_A_Device("serial.device", 0L, &SerReq, 0L, SER_LEN);
...
```

If another program already has access to the serial device, you won't be able to access it at that time. The serial device offers the option of sharing the device between users:

```
struct IOExtSer *SerReq = 0L;
#define SER_LEN (ULONG) sizeof(struct IOExtSer)
VOID *GetDeviceBlock();
...
   SerReq = (struct IOExtSer *) GetDeviceBlock(SER_LEN);
   SerReq->io_SerFlags = (UBYTE) SERB_SHARED;    Open_A_Device
("serial.device", 0L, &SerReq, 0L, 0L);
...
```

SERB_SHARED has the value 32 (like PARB_SHARED for the parallel device). Be sure that you declare the GetDeviceBlock() as a function with a pointer as the return value. Otherwise the result executes an extension of a long word (ext.1 d0). This interrupts the serial device.

4.4.2 Reading and writing serial device data

The standard commands CMD_READ and CMD_WRITE allow you to read and write through the serial device:

```
/*****************************************************************
*                       Ser_Support.c                            *
*                        August 1988                             *
*                     (c) Bruno Jennrich                         *
* Compile-Info:                                                  *
* cc Ser_Support.c                                               *
```

```
*****************************************************************/
#include "exec/types.h"
#include "exec/io.h"
#include "devices/serial.h"
/****************************************************************
*                          Serial_Read()         (Ser_Support)*
* Function: Read data                                          *
*--------------------------------------------------------------*
* Input - Parameter:                                           *
* SerReq: Device-Block                                         *
* Data:   Data buffer                                          *
* Len:    Amount of data to be read                            *
*****************************************************************/
VOID Serial_Read (SerReq,Data,Len)
struct IOExtSer  *SerReq;
APTR                    Data;
ULONG                        Len;
{
   SerReq->IOSer.io_Data = Data;
   SerReq->IOSer.io_Length = Len;
   Do_Command (SerReq,(UWORD) CMD_READ);
}
/****************************************************************
*                          Serial_Write()        (Ser_Support)*
* Function: Write data                                         *
*--------------------------------------------------------------*
* Input - Parameter:                                           *
* SerReq: Device-Block                                         *
* Data:   Data to be written                                   *
* Len:    Amount of data to be written                         *
*****************************************************************/
VOID Serial_Write (SerReq,Data,Len)
struct IOExtSer   *SerReq;
APTR                    Data;
ULONG                        Len;
{
   SerReq->IOSer.io_Data = Data;
   SerReq->IOSer.io_Length = Len;
   Do_Command (SerReq,(UWORD) CMD_WRITE);
}
```

These two commands require the address of the data to be sent, or the address of the buffer to which the data should be written, as well as the number of bytes to be transferred. If you enter a value of -1 for `Len`, the serial device writes data until it encounters a null byte. The serial device writes this null byte and stops writing data.

When reading you must determine whether `io_TermArray` is used. The serial device reads data until a character from the `TermArray` is received. When the serial device encounters a null byte during reading, the reading process stops.

Note: The above functions use the `DoIO()` command for command execution. It could be as important to work with `SendIO()`, `CheckIO()` and `WaitIO()` to implement longer transfer time, when much data is sent.

4.4.3 Serial device parameters

When reading about serial interfaces, you'll see many buzz phrases like transfer protocol, word length, baud rate and stop bits. The serial device allows you to set your own serial interface parameters. Let's take a closer look at these parameters.

Transfer protocols

Serial transfer reads and writes data one bit at a time. An error occurring during this transfer is quite possible. In parallel transfer (one byte at a time instead of one bit at a time), the odds of errors increase eight times. Serial transfer offers the programmer many different *transfer protocols*. These protocols allow a "re-take" of an incorrectly transferred byte.

The serial device currently supports the XOn/XOff transfer protocol. XOn/XOff is the default protocol (after `OpenDevice()`) unless the `SERB_XDISABLED` bit (bit 7) is set (bit 7 = 128). It is turned off when `SerReq->io_SerFlags = SERB_XDISABLED`. Control characters, which allow control over XOn/XOff transfer, are determined by `SerReq->_io_CtlChar`. Like `TermArray`, this element consists of a ULONG which can read the ASCII codes from characters: Bits 31-24 test the XOn character, and bits 23-16 determine the XOff character. Bits 15-8 should take the INQ character, while bits 7-0 should be used for the ACK character. The INQ and ACK (handshaking) are not currently supported by the serial device.

Bits

The number of bits per byte that you want to send or receive also directly affect the transfer protocol. You have the option of sending 7 or 8 bits (`SerReq->io_WriteLen`) or receiving 7 or 8 bits (`SerReq->io_ReadLen`). A stop bit follows the seventh or eight bit. This stop bit marks the end of the transferred value. Seven bits are most often used to send and receive ASCII codes. The ASCII codes here have the values 0 to 0xf. You can increase the number of stop bits to 2 to further ensure data security when you are sending seven bits. (`SerReq->io_StopBit = (BYTE) 2`).

Baud rate

In addition to the transfer protocol we must determine the speed at which the data should be transferred. The *baud rate* specifies the number of bits transferred per second. The Amiga can handle serial transfer from 112 baud (bits per second) to 292,000 baud (bits per second). Insert your baud rate in `SerReq->io_Baud`. The normal minimum setting is 110 baud; the Amiga can only process a minimum of 112 baud because of its hardware design.

Break When you want to interrupt the data transfer you must send a break signal. The break signal ensures that all of the connections are set to zero for a specific amount of time. The time that the connections should be in the low condition can be specified in microseconds in `SerReq->io_BrkTime`. The `SDCMD_BREAK` command sends the break signal. You must be sure that the same parameters exist on both the receiver's side and the sender's side, otherwise the transfer will not interact.

Buffers The Amiga also manages some software based parameters. The serial device also controls one of the Amiga's own read buffers. Normally this buffer is 512 bytes. If you need a larger buffer, you can specify the new buffer length in `SerReq->io_RBufLen`. You must then execute `SDCMD_SETPARAMS`. First the new buffer is allocated and the data that was previously stored in the old buffer is lost. All of the parameters that were changed are first given to the serial device after `SDCMD_SETPARAMS`.

Terminators It is the same with a change of the terminators. You simply specify the eight (or less) new terminators that end a read command (Len = -1) and set the `EOFMODE` flag in `SerReq->io_SerFlag`. The terminators are used after `SDCMD_SETPARAMS`. You should make sure that the only parameter change during a read or write operation is the change to the `SERB_XDISABLED` parameter. Any other changes abort transfer with an error.

Flags The serial device includes a set of flags that can control data transfer. Here are the flags and what they do:

SERB_PARTY_ON (1):

Checks parity of bits received.

SERB_PARTY_ODD (2):

Checks for odd parity (total of the digits = 1). If this bit is clear, even parity is used.

SERB_7WIRE (4):

When set before `OpenDevice()`, seven-wire communication becomes active. Normal data transfer uses three lines:

TXD (TRANSMIT DATA)
RXD (RECEIVE DATA)
GND (GROUND)

Seven-wire handshaking adds four wires for a total of seven lines:

TXD (TRANSMIT DATA)
RXD (RECEIVE DATA)
GND (GROUND)
RTS (REQUEST TO SEND)
CTS (CLEAR TO SEND)
DSR (DATA SET READY)
DCD (DATA CARRIER DETECT).

SERB_QUEUEDBRK (8):

Controls enqueued break commands. The queue is an area of memory into which serial output is stored and transmitted. The queue operates on a first in, first out (or FIFO) basis. If the SERB_QUEUEDBRK bit is set, the system executes the current serial output commands sequentially, ending with the break command (SDCMD_BREAK). If this bit is cleared (default state), the break has first priority over any other serial output waiting in the queue. Once the break command executes, the interrupted request continues execution, unless the user aborts the request.This flag may be set with SDCMD_SETPARAMS.

SERB_RAD_BOOGIE (16):

Controls high-speed mode. If this bit is set, parity check is disabled, XOn/XOff protocol is disabled, SERB_XDISABLED is set and the system consistently sends eight-bit data. Some external devices such as MIDI equipment require high-speed data transfer.

SERB_SHARED (32):

Controls sharing the serial interface with other users. This flag can only be set before OpenDevice(), or Open_A_Device().

SERB_EOFMODE (64):

Controls io_TermArray and IORequest usage. Setting this flag instructs the serial device to verify characters against io_TermArray, and instructs the serial device to end IORequest as soon as the device detects and end of file character. This flag may be set without SDCMD_SETPARAMS to activate and deactivate the established terminators.

SERB_XDISABLED (128):

Disables XOn/XOff protocol. It is enabled after OpenDevice().

SDCMD_SETPARAMS is consistently used to indicate a parameter change for the serial device.

4.4.4 Reading serial interface status

The following command sequence displays the status of the serial device:

```
Do_Command(SerReq, (UWORD)SDCMD_QUERY);
Status = SerReq->io_Status;
```

Calling `Do_Command (SerReq, (UWORD) SDCMD_QUERY)` returns the status of the serial interface in `SerReq->io_Status`:

```
Bit 0        = 0: BUSY
             = 1: no transfer
Bit 1        = 0: Paper out
             = 1: Paper is present
Bit 2        = 0: ONLINE
             = 1: OFFLINE
Bit 3        = 0: Data Set Ready
             = 1: No data
Bit 4        = 0: Clear To Send
             = 1: Not clear
Bit 5        = 0: Carrier Detect (carrier signal present)
             = 1: No carrier
Bit 6        = 0: Ready To Send
             = 1: Not ready
Bit 7        = 0: Data Terminal Ready
             = 1: Not ready
Bit 8        = 1: Read Buffer Overrun (Read buffer full)
             = 0: No overrun
Bit 9        = 1: Break Sent
             = 0: No break
Bit 10       = 1: Break received
             = 0: No break
Bit 11       = 1: Transmit XOFFed (xOff sent)
             = 0: No XOFF
Bit 12       = 1: Received XOFFed (xOff received)
             = 0: No XOFF
Bits 13-15     Unused
```

In addition to checking the status word, you have the option of checking the variable `SerReq->IOSer.io_Flags`. The most important conditions are saved here:

IOSERF_OVERRUN	(1)	Read buffer overrun
IOSERF_WRITEBREAK	(2)	Break sent
IOSERF_READBREAK	(4)	Break received
IOSERF_XOFFWRITE	(8)	XOff written
IOSERF_XOFFREAD	(16)	XOff received
IOSERF_ACTIVE	(32)	Read or write access executing
IOSERF_ABORT	(32)	`AbortIO()` executed

IOSERF_QUEUED	(64)	Read/write announced but not executed (another read/write already active)
IOSERF_BUFREAD	(128)	Data read from the internal buffer

As you see, bit four (32) appears twice. This may or may not be an error, which makes it hard to determine which bit stands for which condition. Avoid checking the `io_Flags` variable, and go directly over the `SDCMD_QUERY` and `io_Status` instead.

4.4.5 Serial device error messages

Serial device errors occur easily during the initial phases of developing serial access programs. The following list describes the standard serial device errors you may encounter:

SerErr_DevBusy	(1)	Reading or writing in process. SETPARAMS cannot be executed
SerErr_BaudMismatch	(2)	Baud rates of sender and receiver do not match
SerErr_InvBaud	(3)	Baud rate less than 112 and more than 292,000 baud
SerErr_BuffErr	(4)	Internal buffer size is less than 512 bytes or too large (insufficient memory)
SerErr_InvParam	(5)	Parameter change not allowed
SerErr_LineErr	(6)	Transfer error (possibly defective connection)
SerErr_NotOpen	(7)	Cannot find serial.device
SerErr_PortReset	(8)	Interface reset
SerErr_ParityErr	(9)	Parity error in transfer
SerErr_InitErr	(10)	Device initialization error
SerErr_TimeErr	(11)	Error in io_BrkTime
SerErr_BufOverflow	(12)	Read buffer overflow
SerErr_NoDsr	(13)	No Data Set Ready signal
SerErr_NoCTS	(14)	No Clear To Send signal
SerErr_DetectedBreak	(15)	Break detected

4.4.6 Serial port pins

Pin	A500	A1000	A2000	
1	GND	GND	GND	(Ground)
2	TXD	TXD	TXD	(Transmit Data)
3	RXD	RXD	RXD	(Receive Data)
4	RTS	RTS	RTS	(Request To Send)
5	CTS	CTS	CTS	(Clear To Send)
6	DSR	DSR	DSR	(Data Set Ready)
7	GND	GND	GND	(Ground)
8	DCD	DCD	DCD	(Data Carrier Detect [receive carrier signal])
9	+12v	NC	+12v	
10	-12v	———	-12v	
11	AUDO	———	AUDO	(Audio Output)
12	———	———	———	
13	———	———	———	
14	———	-5v	———	
15	———	AUDO	———	(Audio Output)
16	———	AUDI	———	(Audio Input)
17	———	EB	———	(716 KHz Takt)
18	AUDI	INT2*	AUDI	(External interrupt [IRQ])
19	———	———	———	
20	DTR	DTR	DTR	(Data Terminal Ready)
21	———	+5v	———	
22	RI	———	RI	(Ring Indicator)
23	———	+12v	———	
24	———	C2*	———	(3.58 MHz)
25	———	RESB*	———	(Buffered reset)

To conclude here is a layout for a null modem cable to connect two computers over the serial interface, and a short application for using the null modem cable.

Null modem cable

The connector used with Amiga serial port is a DB25 (25-pin) connector. The Amiga 1000 uses a DB25 male connector, while the Amiga 500 and 2000 accept a DB25 female connector. This is important when you go into an electronics store to get parts for the null modem cable. The RS-232 connection is crossed in the null modem cable, as shown in the following table:

Null modem cable connections

Pin	Computer A	Computer B	Pin
1	GND	GND	1
2	TXD	RXD	3
3	RXD	TXD	2
4	RTS	DCD	8
5	CTS	DCD	8
6	DSR	DTR	20
20	DTR	DSR	6
8	DCD	RTS	4
7	GND	GND	7
8	DCD	CTS	5

Connect pin 2 of one connector with pin 3 of the other connector, and so on.

4.4.7 A serial device application

The following program transfers data between two computers using the null modem cable described in Section 4.4.6.

```
/*****************************************************************
*                            Ser.c                               *
*                         August 1988                            *
*                     (c) Bruno Jennrich                         *
*                                                                *
* Function: Access serial interface                              *
*****************************************************************/

/*****************************************************************
* Compile-Info:                                                  *
*                                                                *
* cc Ser                                                         *
* ln Ser.o Ser_Support.o -Devs_Support.o -lc                     *
*****************************************************************/

#include "exec/types.h"
#include "exec/io.h"
#include "devices/serial.h"

struct IOExtSer *SerReq;
#define SER_LEN (ULONG) sizeof (struct IOExtSer)


/*****************************************************************
*                         CloseIt()                        (User)*
*                                                                *
* Function: In case of error, close everything                   *
*---------------------------------------------------------------*
```

```
* Input - Parameter:                                             *
*                                                                *
* String: Error-Message                                          *
*****************************************************************/

VOID CloseIt (String)
char        *String;
{
   UWORD i;
   UWORD *dff180 = (UWORD *)0xdff180;
   UWORD Error = 0;

   if (strlen (String) > 01)
   {
      for (i=0;i<0xffff;i++)  *dff180 = i;
      puts (String);
      Error = 10;
   }
   if (SerReq != 01) Close_A_Device (SerReq);
   exit (Error);
}

/*****************************************************************
*                           main()                               *
*                                                                *
*----------------------------------------------------------------*
* Input - Parameter:                                             *
*                                                                *
* When argc  > 1 => read data                                    *
* When argc == 0 0> write data                                   *
*****************************************************************/

main (argc,argv)
UWORD argc;
BYTE      *argv[];
{
   BYTE Buffer[256];

   Open_A_Device ("serial.device",01,&SerReq,01,SER_LEN);

   if (argc > 1)
   {
      Serial_Read (SerReq,Buffer,-1);
      printf ("%s \n",Buffer);
   }
   else
      Serial_Write (SerReq,"HELLO",-1);
   Close_A_Device (SerReq);
}
```

Call the program with any command parameter (e.g., `Ser x`) on the receiving Amiga. This Amiga waits until data is received. Remove the disk from the drive and place it in the sending Amiga. Start the program without command parameters (enter `Ser`) and watch the result: The text sent appears on the receiving computer's screen.

4.5 The printer device

The printer device allows access to a printer. The printer device has three different types of access:

Text	`CMD_WRITE` (3) & `PRD_RAWWRITE` (9)
Command	`PRD_PRTCOMMAND` (10)
Hardcopy	`PRD_DUMPRPORT` (11)

These three task regions of the printer device use three different device blocks. The printer commands and hardcopy access have their own special device blocks:

```
Offset    Structure
------    --------
          struct IOPrtCmdReq /* Command Request */
          {
0  0x00      struct Message io_Message;
20 0x14      struct Device *io_Device;
24 0x18      struct Unit   *io_Unit;
28 0x1c      UWORD          io_Command;    /* PRD_PRTCOMMAND */
30 0x1e      UBYTE          io_Flags;
31 0x1f      BYTE           io_Error;
32 0x20      UWORD          io_PrtCommand; /* printer command */
34 0x22      UBYTE          io_Parm0;      /* Parameter */
35 0x23      UBYTE          io_Parm1;
36 0x24      UBYTE          io_Parm2;
37 0x25      UBYTE          io_Parm3;
38 0x26   } /* defined in "devices/printer.h" */
```

```
Offset    Structure
------    --------
          struct IODRPReq /* DumpRastPort Request */
          {
0  0x00      struct Message   io_Message;
20 0x14      struct Device   *io_Device;
24 0x18      struct Unit     *io_Unit;
28 0x1c      UWORD            io_Command;    /* PRD_PRTCOMMAND */
30 0x1e      UBYTE            io_Flags;
31 0x1f      BYTE             io_Error;
32 0x20      struct RastPort *io_RastPort; /* Graphic RastPort*/
36 0x24      struct ColorMap *io_ColorMap; /* color table */
40 0x28      ULONG            io_Modes;    /* ViewPort Modes */
44 0x2c      UWORD            io_SrcX;     /* Start point */
46 0x2e      UWORD            io_SrcY;
48 0x30      UWORD            io_SrcWidth; /* width */
50 0x32      UWORD            io_SrcHeight;/* Height */
52 0x34      LONG             io_DestCols; /* print width */
56 0x38      LONG             io_DestRows; /* print height */
60 0x3c      UWORD            io_Special;  /* Special Flags */
62 0x3e   } /* defined in "devices/printer.h" */
```

The following routine allows open access to all three device blocks through the `Normal`, `DumpRastPort` and `Command` pointers:

```
struct IODRPReq    *PrtPtr = 0L; /* Dummy pointer */
struct IOStdReq    *Normal;
struct IODRPReq    *DumpRastPort;
struct IOPrtCmdReq *Command;
#define PRT_LEN (ULONG) sizeof (struct IODRPReq)/*larger block*/
   Open_A_Device ("printer.device", 0L, &PrtPtr, 0L, PRT_LEN);
   Normal       = (struct IOStdReq *)PrtPtr;
   DumpRastPort = (struct IODRPReq *)PrtPtr;
   Command      = (struct IOPrtCmdReq *)PrtPtr;
```

4.5.1 Printing escape sequences

You can send your texts to the printer with the `PRD_RAWWRITE` command. The escape sequences are not replaced (see `CMD_WRITE`). What you specified as the output string, is also output:

```
/*****************************************************************
*                      Printer_RawWrite()       (Printer_Support)*
*                                                                *
* Function: Display data                                         *
*----------------------------------------------------------------*
* Input - Parameter:                                             *
*                                                                *
* PrtReq: Device-Block (Normal)                                  *
* Data:   String to be displayed                                 *
* Len:    Number of characters to be displayed                   *
*****************************************************************/

Printer_RawWrite (PrtReq,Data,Len)
struct IOStdReq  *PrtReq;
APTR                      Data;
ULONG                         Len;
{
   PrtReq->io_Data   = Data;
   PrtReq->io_Length = Len;
   Do_Command (PrtReq,(UWORD) PRD_RAWWRITE);
}
```

4.5.2 Amiga printer escape sequences

The Amiga can take a standard set of printer escape sequences and translate them for most printers, using the available printer drivers. The following table lists the escape sequences that the Amiga understands; their command numbers; the standard printer escape sequences; and their meanings.

Printer Escape Sequences

Command name	Command number	Escape sequence	Meaning
aRIS	0L	ESCc	Reset
aRIN	1L	ESC#1	Initializing
aIND	2L	ESCD	Linefeed
aNEL	3L	ESCE	LF = CR+LF
aRI	4L	ESCM	Reverse linefeed
aSGR0	5L	ESC[0m	Normal character set
aSGR3	6L	ESC[3m	Italics on
aSGR23	7L	ESC[23m	Italics off
aSGR4	8L	ESC[4m	Underline on
aSGR24	9L	ESC[24m	Underline off
aSGR1	10L	ESC[1m	Bold on
aSGR22	11L	ESC[22m	Bold off
aSFC	12L	ESC[30m-ESC[39m	Set foreground color
aSBC	13L	ESC[40m-ESC49m	Set background color
aSHORP0	14L	ESC[0w	Normal type
aSHORP2	15L	ESC[2w	Elite on
aSHORP1	16L	ESC[1w	Elite off
aSHORP3	18L	ESC[3w	Condensed off
aSHORP6	19L	ESC[6w	Expanded print on
aSHORP5	20L	ESC[5w	Expanded print off
aDEN6	21L	ESC[6"z	Shaded print on
aDEN5	22L	ESC[5"z	Shaded print off
aDEN4	23L	ESC[4"z	Double-strike on
aDEN3	24L	ESC[3"z	Double-strike off
aDEN2	25L	ESC[2"z	NLQ on
aDEN1	26L	ESC[1"z	NLQ off
aSUS2	27L	ESC[2v	Superscript on
aSUS1	28L	ESC[1v	Superscript off
aPLU	32L	ESCL	Superscript (half step)
aPLD	33L	ESCK	Subscript (half step)
aFNT0	34L	ESC(B	US character set
aFNT1	35L	ESC(R	French character set
aFNT2	36L	ESC(K	German character set
aFNT3	37L	ESC(A	English character set
aFNT4	38L	ESC(E	Danish character set 1
aFNT5	39L	ESC(H	Swedish character set
aFNT6	40L	ESC(Y	Italian character set
aFNT7	41L	ESC(Z	Spanish character set

Printer Escape Sequences

Command name	Command number	Escape sequence	Meaning
aFNT8	42L	ESC(J	Japanese character set
aFNT9	43L	ESC(6	Norwegian character set
aFNT10	44L	ESC(C	Danish character set 2
aPROP2	45L	ESC[2p	Proportional text on
aPROP0	47L	ESC[0p	Proportional text off
aTSS	48L	ESC[n E	Proportional spaces=n
aJFY5	49L	ESC[5 F	Left justification
aJFY7	50L	ESC[7 F	Right justification
aJFY6	51L	ESC[6 F	Block characters on
aJFY3	53L	ESC[3 F	Adjust character width
aJFY1	54L	ESC[1 F	Centering
aVERP0	55L	ESC[0z	Line spacing 1/8"
aVERP1	56L	ESC[1z	Line spacing 1/6"
aSLPP	57L	ESC[nt	Set page length (n)
aPERF	58L	ESC[nq	Page break (n>0)
aPERF0	59L	ESC[0q	Page break
aLMS	60L	ESC#9	Set left margin
aRMS	61L	ESC#0	Set right margin
aTMS	62L	ESC#8	Set page header
aBMS	63L	ESC#2	Set page footer
aSTBM	64L	ESC[Pn1r ESC[Pn2r	Set top (n1) and bottom(n2) margins
aSLRM	65L	ESC[Pn1s ESC[Pn2s	Set left (n1) and right (n2) margins
aCAM	66L	ESC#3	Clear all margins
aHTS	67L	ESCH	Horizontal tabs
aVTS	68L	ESCJ	Vertical tabs
aTBC0	69L	ESC[0g	Clear horizontal tab
aTBC3	70L	ESC[3g	Clear all horiz. tabs
aTBC1	71L	ESC[1g	Clear vertical tab
aTBC4	72L	ESC[4g	Clear all vertical tabs
aTBCALL	73L	ESC#4	Clear all tabs
aTBSALL	74L	ESC#5	Set default tabs
aEXTEND	75L	ESC[Pn"x	Extended font

You can see the advantage of using this table. If someone wants to write a text that contains underlined superscripts, the word processor only needs to send the command "ESC[4m" for underline and "ESC[2v" for superscripts. It's the same, no matter what printer you're using. The corresponding printer driver contains a similar table that has the printer specific escape sequences.

The eighth entry of the sequence in this table is "ESC-1" (Star NL-10). The printer device replaces "ESC[4m" with "ESC-1" and the following text is underlined. This functions only when you use the CMD_WRITE command instead of the PRD_RAWWRITE command:

```
/*****************************************************************
*                         Printer_Write()        (Printer_Support)*
*                                                                 *
* Function: output data (Convert Escape-Sequences)                *
*-----------------------------------------------------------------*
* Input - Parameter:                                              *
*                                                                 *
* PrtReq: Device-Block (Normal)                                   *
* Data:   String to be output                                     *
* Len:    Number of characters to be output                       *
*****************************************************************/

Printer_Write   (PrtReq,Data,Len)
struct IOStdReq *PrtReq;
APTR                    Data;
ULONG                       Len;
{
   PrtReq->io_Data   = Data;
   PrtReq->io_Length = Len;
   Do_Command (PrtReq, (UWORD) CMD_WRITE);
}
```

There is no substitution with the RAWWRITE command.

4.5.3 Printer commands

You can use these escape sequences as printer commands. When escape sequences contain only parameters (e.g., setting the left and right margins), no simple substitution can be made. A routine in the printer driver processes the irreplaceable escape sequences. This routine can be accessed directly with the command PRD_PRT COMMAND:

```
/*****************************************************************
*                         Printer_Command()      (Printer_Support)*
*                                                                 *
* Function: Execute printer command                               *
*-----------------------------------------------------------------*
* Input - Parameter:                                              *
*                                                                 *
* PrtReq:  Device-Block                                           *
* Command: command                                                *
* P1-P4:   Parameter                                              *
*****************************************************************/

Printer_Command    (PrtReq,Command,P0,P1,P2,P3)
struct IOPrtCmdReq *PrtReq;
UWORD                      Command;
UBYTE                              P0,P1,P2,P3;
```

```
{
   PrtReq->io_PrtCommand = Command;
   PrtReq->io_Parm0 = P0;
   PrtReq->io_Parm1 = P1;
   PrtReq->io_Parm2 = P2;
   PrtReq->io_Parm3 = P3;
   Do_Command (PrtReq, (UWORD) PRD_PRTCOMMAND);
}
```

Here is a short example. To establish the left and right margins, you can call:

```
Printer_Command (PrtReq, 651, (BYTE)2, (BYTE)78, (BYTE)0,
(BYTE)0);
```

You can also replace the number 651 with the command name aSLRM. The Parm0-Parm4 variables specify the printer driver routines.

4.5.4 Hardcopy

The PRD_DUMPRPORT command provides the developer with an easy method of printing a screen to the printer.

There are a few parameters you must provide when accessing PRD_DUMPPORT:

1.) The RastPort which contains the graphic to be printed.
2.) The ColorMap which contains the graphic's colors (this is especially important when using color printers).
3.) The ViewPort mode variables so that the printer knows the graphic's current display mode (HI-RES, LACE, HAM, etc.).
4.) The upper left corner of the area to be printed (SrcX, SrcY).
5.) The height and width of the area to be printed (SrcHeight, SrcWidth). This allows you to print any rectangular section of the screen (Src = Source = <=> RastPort).
6.) The size of the hardcopy as it should appear on the printer (DestRows, DestCols) (Dest = Destination <=> Printer).
7.) The io_Special flag should be set to zero.

DestRows and DestCols

The DestRows and DestCols parameters can be set in a number of ways. Here are some examples.

```
DestCols>0
DestRows>0
```

The above configuration prints a graphic `DestRows` rows high and `DestCols` columns wide.

```
DestCols=0
DestRows>0
```

The above configuration prints a graphic as wide as the paper and `DestRows` rows high.

```
DestCols=0
DestRows=0
```

The above configuration prints a graphic as wide as the paper and as high as the paper.

```
DestCols>0
DestRows=0
```

The above configuration prints a graphic `DestCols` columns wide, with the height proportional to the width.

```
DestCols<0
DestRows>0
```

The above configuration enlarges or reduces the size of the printed graphic. The equation for computing this is as follows:

```
  |DestCols|
 ----------- = enlargement factor
  DestRows
```

For example, if `DestCols` has a value of -1 and `DestRows` has a value of 4, the enlargement/reduction factor is equal to 1/4. This value only applies to `DestCols` and `DestRows` if `io_Special` is zero. The following lines describe the individual `Special_Flags` and their tasks in the printing process:

SPECIAL_MILCOLS 0x001L	`DestCols` given in 1/1000
SPECIAL_MILROWS 0x002L	`DestRows` given in 1/1000
SPECIAL_FULLCOLS 0x004L	`DestCols` set at maximum
SPECIAL_FULLROWS 0x008L	`DestRows` set at maximum
SPECIAL_FRACCOLS 0x010L	Width = maximum/`DestRows`
SPECIAL_FRACROWS 0x020L	Height = maximum/`DestRows`
SPECIAL_ASPECT 0x080L	If set, either height or width changes to preserve page set up
SPECIAL_DENSITY1 0x100L	Print density (1=low; 4=high)
SPECIAL_DENSITY2 0x200L	
SPECIAL_DENSITY3 0x300L	
SPECIAL_DENSITY4 0x400L	
SPECIAL_CENTER 0x040L	Center graphic

Maximum printable area

The printer driver contains the width and height of the maximum printable area. You get page setups from these two maxima (MaximaX/MaximaY), reserved through Special_Aspect.

```
/**************************************************************
*                          Printer_Dump()    (Printer_Support)*
*                                                             *
* Function: Hardcopy                                          *
*-------------------------------------------------------------*
* Input - Parameter:                                          *
*                                                             *
* PrtPtr:    Device-Block                                     *
* RastPort:  RastPort of the graphic to be printed            *
* ColorMap:  ColorMap contains the actual colors              *
* Modes:     Display modes                                    *
* SrcX,SrcY: Top left corner of graphic to be printed         *
* SrcWidth,                                                   *
* SrcHeight: Width and height of the graphic to be printed    *
* DestCols:  Number of columns (Printer)                      *
* DestRows:  Number of lines   (Printer)                      *
* Special:   Special-Flags                                    *
**************************************************************/

VOID Printer_Dump (PrtPtr,RastPort,ColorMap,Modes,SrcX,SrcY,
                   SrcWidth,SrcHeight,DestCols,DestRows,Special)
struct IODRPReq   *PrtPtr;
struct RastPort          *RastPort;
struct ColorMap                   *ColorMap;
ULONG                                       Modes;
UWORD                                             SrcX,SrcY;
UWORD              SrcWidth,SrcHeight;
LONG                                 DestCols,DestRows;
UWORD                                                   Special;
{
   PrtPtr->io_RastPort  = RastPort;
   PrtPtr->io_ColorMap  = ColorMap;
   PrtPtr->io_Modes     = Modes;        /* Viewmodes */
   PrtPtr->io_SrcX      = SrcX;         /* Start point */
   PrtPtr->io_SrcY      = SrcY;
   PrtPtr->io_SrcWidth  = SrcWidth;     /* Width */
   PrtPtr->io_SrcHeight = SrcHeight;    /* Height */
   PrtPtr->io_DestCols  = DestCols;     /* Print width */
   PrtPtr->io_DestRows  = DestRows;     /* Print height */
   PrtPtr->io_Special   = Special;      /* Special-Flags */

   Do_Command (PrtPtr, (UWORD) PRD_DUMPRPORT);
}
```

The following program uses the Dump routine to print a section of the current window. You must select the section using the mouse. Combine the previous printer support routines to make the Printer_Support.c file. Don't forget the include files in the Printer_suppport.c file.

```
/****************************************************************
*                            Prt.c                              *
*                         August 1988                           *
*                     (c) Bruno Jennrich                        *
*                                                               *
* Function: Hardcopy of the current Window                      *
****************************************************************/
/****************************************************************
* Compile-Info:                                                 *
*                                                               *
* cc Prt                                                        *
* ln Prt.o Printer_Support.o Devs_Support.o -lc                 *
****************************************************************/
#include "exec/types.h"
#include "exec/io.h"
#include "devices/printer.h"
#include "intuition/intuitionbase.h"
#include "intuition/intuition.h"
#include "graphics/gfxbase.h"
#include "graphics/view.h"
union PrinterIO
{                                   /* Printer Blocke */
   struct IOStdReq    Normal;
   struct IODRPReq    DumpRastPort;
   struct IOPrtCmdReq Command;
};
union PrinterIO     PrtReq;
struct IODRPReq    *PrtPtr = 0l;
struct IOStdReq    *Normal;
struct IODRPReq    *DumpRastPort;
struct IOPrtCmdReq *Command;

#define PRT_LEN (ULONG) sizeof (struct IODRPReq)

struct IntuitionBase *IntuitionBase = 0L;
struct Window        *Window = 0L;
struct GfxBase       *GfxBase = 0L;

VOID *OpenLibrary();
/****************************************************************
*                        CloseIt()                       (User)*
*                                                               *
* Function: In case of error close everything                   *
*---------------------------------------------------------------*
* Input-Parameter:                                              *
*                                                               *
* String: Error-Message                                         *
****************************************************************/
VOID CloseIt (String)
char          *String;
{
   UWORD i;
   UWORD *dff180 = (UWORD *)0xdff180;
   UWORD Error = 0;

   if (strlen (String) > 0l)
   {
      for (i=0;i<0xffff;i++)  *dff180 = i;
      puts (String);
      Error = 10;
   }
   if (PrtPtr != 0l)       Close_A_Device (PrtPtr);
```

```
   if (IntuitionBase == 0l) CloseLibrary (IntuitionBase);
   if (GfxBase == 0l)       CloseLibrary (GfxBase);
   exit (Error);
}
/****************************************************************
*                       Mark_PrintArea()                 (User)*
*                                                               *
* Function: Choose section for hardcopy                         *
*---------------------------------------------------------------*
* Input-Parameter:                                              *
*                                                               *
* Window: Address of the window to be printed                   *
* x1,y1,x2,y2: later contains the upper left and lower          *
*              corner of the area to be printed                 *
****************************************************************/
Mark_PrintArea (Window,x1, y1, x2, y2)
struct Window  *Window;
ULONG               *x1,*y1,*x2,*y2;
{
   UBYTE *LeftMouse = (UBYTE *)0xbfe001;
   ULONG xold,yold;

   *x1 = (ULONG) 0;
   *y1 = (ULONG) 0;
   *x2 = (ULONG) 0;
   *y2 = (ULONG) 0;

   SetDrMd (Window->RPort,COMPLEMENT);

   while ((*LeftMouse & (UBYTE)0x40) == (UBYTE)0x40);

   *x1 = (ULONG)Window->MouseX;
   *y1 = (ULONG)Window->MouseY;

   xold = *x1;
   yold = *y1;

   Move (Window->RPort,*x1,*y1);          /* first rectangle */
   Draw (Window->RPort,xold,*y1);
   Draw (Window->RPort,xold,yold);
   Draw (Window->RPort,*x1,yold);
   Draw (Window->RPort,*x1,*y1);

   while ((*LeftMouse & (UBYTE)0x40) == (UBYTE)0x0)
   {
      *x2 = (ULONG)Window->MouseX;
      *y2 = (ULONG)Window->MouseY;

      if ((*x2 != xold) || (*y2 != yold))
      {
         Move (Window->RPort,*x1,*y1);    /* Rubberband-
Rectangle */
         Draw (Window->RPort,xold,*y1);
         Draw (Window->RPort,xold,yold);
         Draw (Window->RPort,*x1,yold);
         Draw (Window->RPort,*x1,*y1);

         Move (Window->RPort,*x1,*y1);
         Draw (Window->RPort,*x2,*y1);
         Draw (Window->RPort,*x2,*y2);
         Draw (Window->RPort,*x1,*y2);
         Draw (Window->RPort,*x1,*y1);
```

```
            xold = *x2;
            yold = *y2;
         }
      }
      Move (Window->RPort,*x1,*y1);        /* erase rectangle */
      Draw (Window->RPort,xold,*y1);
      Draw (Window->RPort,xold,yold);
      Draw (Window->RPort,*x1,yold);
      Draw (Window->RPort,*x1,*y1);

      SetDrMd (Window->RPort,JAM2);
      if (*x1 > *x2)
      {
         xold = *x1;
         *x1 = *x2;
         *x2 = xold;
      }
      if (*y1 > *y2)
      {
         yold = *y1;
         *y1 = *y2;
         *y2 = yold;
      }
}
/************************************************************
*                        main()                      (User)*
*                                                           *
* May have to change SPECIAL_DENSITY1 depending on quality  *
* supported by your printer                                 *
************************************************************/
main()
{
   ULONG x1,y1,x2,y2;
   if ((IntuitionBase = (struct IntuitionBase *)
        OpenLibrary ("intuition.library",0L)) == (struct
IntuitionBase *) 0l)
      CloseIt ("No Intuition !!!");
   if ((GfxBase = (struct GfxBase *)
        OpenLibrary ("graphics.library",0L)) == (struct GfxBase
*) 0l)
      CloseIt ("No Graphics !!!");
   Window = IntuitionBase->ActiveWindow;
   Mark_PrintArea (Window,&x1,&y1,&x2,&y2);
   Open_A_Device ("printer.device",0L,&PrtPtr,0L,PRT_LEN);
   Normal       = (struct IOStdReq *)PrtPtr;
   DumpRastPort = (struct IODRPReq *)PrtPtr;
   Command      = (struct IOPrtCmdReq *)PrtPtr;
   Printer_Dump (DumpRastPort,
                 Window->RPort,
                 GfxBase->ActiView->ViewPort->ColorMap,
                 (ULONG)GfxBase->ActiView->ViewPort->Modes,
                 (UWORD)x1,
                 (UWORD)y1,
                 (UWORD)(x2-x1),
                 (UWORD)(y2-y1),
                 (ULONG)(x2-x1),
                 (ULONG)(y2-y1),
                 (UWORD)SPECIAL_DENSITY1);
   CloseLibrary (GfxBase);
   CloseLibrary (IntuitionBase);
   Close_A_Device (PrtPtr);
}
```

4.5.5 Printer device error messages

The printer device contains the following messages:

`#define PDERR_NOERR 0`
No error—everything OK

`#define PDERR_CANCEL 1`
Printer operation interrupted (`AbortIO()`)

`#define PDERR_NOTGRAPHICS 2`
Printer does not support graphics

`#define PDERR_INVERTHAM 3`
You cannot print inverted HAM pictures (Kick1.1 and Kick1.2 only)

`#define PDERR_BADDIMENSION 4`
Invalid print size

`#define PDERR_DIMENSIONOVFLOW 5`
Too large a print size chosen (Kick1.1 and Kick1.2 only)

`#define PDERR_INTERNALMEMORY 6`
Insufficient memory present for internal variables

`#define PDERR_BUFFERMEMORY 7`
Insufficient memory for printer driver to allocate printer buffer

These errors are returned in the `io_Error` variable of the device blocks.

4.5.6 The printer device under Kickstart 1.3

Printer drivers are much faster under Kickstart 1.3. Kickstart 1.3 also provides a new command for the printer device: the `PRD_QUERY` command (`#define PRD_QUERY 12`), which returns the current printer port status. You may remember other forms of this command from the parallel and serial devices. This way the port to which the printer is connected (i.e., Preferences) is the port that is chosen. To get the status, you must enter two `UBYTES` or one `UWORD` and give the address of these words to the `io_Data` pointer:

```
UWORD Status;
PrtReq->io_Data = (APTR) &Status
```

The status stands in this `UWORD` after the call of the `PRD_QUERY` command (`Do_Command(PrtReq, (UWORD) PRD_QUERY)`. The entire `UWORD` is needed for the serial printer, while the parallel printer needs only the lower byte (bits 0-7) of the `UWORD` to reserve the status. With that you know how you must interpret the status returned in `io_Actual`. You can tell if it is being handled as a serial (`io_Actual == 2`) or a parallel printer (`io_Actual == 1`). The sections on the serial and parallel devices indicate the meanings of these bits.

In addition to the new command, there are some new flags:

```
#define SPECIAL_DENSITY5 0x0500
#define SPECIAL_DENSITY6 0x0600
#define SPECIAL_DENSITY5 0x0700
```

You can now select seven print densities instead of four, if the printer supports these.

```
#define SPECIAL_NOFORMFEED 0x0800
```

A page oriented printer (e.g., laser printers or friction feed printers) usually executes a formfeed after printing a page. This ensures that you only print one graphic to a sheet of paper. When you set this flag, no formfeed is executed.

```
#define SPECIAL_TRUSTME 0x1000
```

The printer is usually reset after a hardcopy. To print multiple graphics on the same page, you must instruct the computer not to reset after a hardcopy by setting the `SPECIAL_TRUSTME` flag.

```
#define SPECIAL_NOPRINT 0x2000
```

This flag stops the printer from printing. This may seem silly—why open the printer device and then print nothing? Remember that the printer driver calculates the size of the hardcopy based on the data you placed in `io_DestCols`, `io_YDotsInch`, etc. The size of the hardcopy on the printer then passes to the variables `io_DestCols` and `io_DestRows`. `io_DestCols` and `io_DestRows` contain the number of columns and lines needed to create the hardcopy. Using the `SPECIAL_NOPRINT` flag can then test, for example, if the hardcopy can be printed in the desired size, or if some parameters need changing.

4.6 The keyboard device

The keyboard device allows direct reading of the keyboard. The keyboard device uses the following commands:

KBD_READEVENT	(10)	prepares keyboard input as an input event structure
KBD_READMATRIX	(11)	reads status of all keys (pressed or not pressed)
CMD_CLEAR	(5)	clears keyboard buffer

The keyboard device also features reset handler routines:

KBD_ADDRESETHANDLER	(12)	inserts reset handler
KBD_REMRESETHANDLER	(13)	removes reset handler
KBD_RESETHANDLER-DONE	(14)	informs user that a reset routine has been executed

Of these commands, the only command that currently works is the KBD_READEVENT command. All of the other commands return error messages that say that the command is not implemented (-0xfc). This means that the input task checks this device and does not allow any other users. Regardless, we'll discuss all of the commands.

4.6.1 Opening the keyboard device

The keyboard device uses a standard request block. The following code opens access to the device:

```
struct IOStdReq *KeyRequest = 0L;
#define KEY_LEN (ULONG) sizeof(struct IOStdReq)
...
   Open_A_Device("keyboard.device", 0L, &KeyRequest, 0L,
KEY_LEN);
...
```

4.6.2 Reading the keyboard device

As mentioned above, you have two options for viewing the keyboard status. The first consists of allowing the input device to fill an input structure, which you can examine:

```
/****************************************************************
*                        KeyBoard_ReadEvent()       (Key_Support)*
*                                                               *
* Function: Read KeyBoard-Event                                 *
*---------------------------------------------------------------*
* Input - Parameter:                                            *
*                                                               *
* GamePortRequest:   GamePort-Device-Block                      *
* Type:              Controller-Type                            *
****************************************************************/
VOID KeyBoard_ReadEvent (KeyRequest,InputEvent)
struct IOStdReq         *KeyRequest;
struct InputEvent                  *InputEvent;
{
   KeyRequest->io_Data   = (APTR) InputEvent;
   KeyRequest->io_Length = (ULONG) sizeof (struct InputEvent);
   Do_Command (KeyRequest,(UWORD)KBD_READEVENT);
}
```

The second option consists of reading the status of all of the keys (this second option is currently not implemented, but may be added soon). The address of a byte array is first given to the `io_Data` pointer of the device block. This array will later contain the status of every key on the keyboard. Each bit of this array represents one key (bit = 0 => key not pressed).

The first byte contains the status for the keys coded 0-7, the second contains the status for the keys coded 8-15, and so on. You must specify the number of bytes of the array in `io_Length` to execute the `KDB_READMATRIX` command. Then the array fills with the status of each key.

```
/****************************************************************
*                        KeyBoard_ReadMatrix()      (Key_Support)*
*                                                               *
* Function: Read key statsu                                     *
*---------------------------------------------------------------*
* Input - Parameter:                                            *
*                                                               *
* KeyRequest: Device-Block                                      *
* Array:      Byte-Array for Status                             *
* Len:        Size of arrays                                    *
****************************************************************/
VOID KeyBoard_ReadMatrix (KeyRequest,Array,Len)
struct IOStdReq         *KeyRequest;
APTR                                Array;
```

```
ULONG                                   Len;
{
   KeyRequest->io_Data   = (APTR) Array;
   KeyRequest->io_Length = Len;
   Do_Command (KeyRequest,(UWORD)KBD_READMATRIX);
}
```

4.6.3 Resets through the keyboard device

The keyboard device has provisions for executing system reset routines. These routines execute when the user presses the <Ctrl><left Amiga><right Amiga> keys. These routines may close open files when a reset occurs.

The resets routine

This routine is installed through an interrupt structure whose `is_Code` (program code) and `is_Data` (data) elements are initialized. This interrupt structure is given in the `io_Data` pointer of the device block, where the `io_Length` variable of the value `sizeof (struct interrupt)` is. Then the `KBD_ADDRESETHANDLER` command is called:

```
struct Interrupt OwnReset
...
   OwnReset.is_Code = YourOwnRoutine;
   OwnReset.is_Data = YourOwnData;
    KeyReq->io_Data   = (APTR) OwnReset;
   KeyReq->io_Length = (ULONG) sizeof(struct Interrupt);
   Do_Command(KeyReq, (UWORD) KBD_ADDRESETHANDLER);
```

Unfortunately, this command is not implemented. You may be able to crash the system by accessing this command, which requires powering down on most of the Amiga models.

`KBD_REMRESETHANDLER` works to some extent. With the help of this command the reset handler should be removed shortly before the end of a program. The variables that must be initialized are the same as those used by `KBD_ADDRESETHANDLER`.

The last command, which the keyboard device completely supports at this time, is the `KBD_RESETHANDLERDONE` command. This command must be called after the execution of a reset routine. `KBD_RESETHANDLERDONE` informs the system that the reset handler has been completely processed and the reset can be continued. We don't know if this command functions—we were never able to install a reset handler.

4.6.4 A keyboard device application

The following program reads the input events from the keyboard and displays the key code of the pressed key (`ie_Code`). Combine the two keyboard support routines and the include files listed in the folowing program to make the Key_Support.c file.

```
/****************************************************************
*                          Key.c                                *
*                   (c) Bruno Jennrich                          *
****************************************************************/
/****************************************************************
* Compile-Info:                                                 *
* cc Key.c                                                      *
* ln Key.o Key_Support.o Devs_Support.o -lc                     *
****************************************************************/
#include "exec/types.h"
#include "exec/io.h"
#include "exec/devices.h"
#include "devices/inputevent.h"
#include "devices/keyboard.h"
struct InputEvent Event;
struct IOStdReq *KeyRequest=0l;
#define KEY_LEN (ULONG) sizeof (struct IOStdReq)
/****************************************************************
*                       CloseIt()                       (User)*
* Function: In case of error, close everything                  *
*---------------------------------------------------------------*
* Input - Parameter:                                            *
* String: Error-Message                                         *
****************************************************************/
VOID CloseIt (String)
char         *String;
{
   UWORD i;
   UWORD *dff180 = (UWORD *)0xdff180;
   UWORD Error = 0;
   if (strlen (String) > 0l)
   {
      for (i=0;i<0xffff;i++)  *dff180 = i;
      puts (String);
      Error = 10;
   }
   if (KeyRequest != 0l) Close_A_Device (KeyRequest);
   exit (Error);
}
main()
{
   Open_A_Device ("keyboard.device",0l,&KeyRequest,0l,KEY_LEN);
   do
   {
      KeyBoard_ReadEvent (KeyRequest,&Event);
      printf ("ie_Code: %d\n",Event.ie_Code);
   } while (Event.ie_Code != (UWORD) 0x45); /* Escape */
   Close_A_Device (KeyRequest);
}
```

4.7 The gameport device

The gameport device accesses any connections to the gameports (e.g., mice or joysticks). The gameport device supports the following commands:

GPD_READEVENT	(9)	read controller status
GPD_ASKCTYPE	(10)	read controller type
GPD_SETCTYPE	(11)	set controller type
GPD_ASKTRIGGER	(12)	read announcement status
GPD_SETTRIGGER	(13)	set announcement conditions
CMD_CLEAR	(5)	erase gameport buffer

4.7.1 Opening the gameport device

When opening a device you must determine which gameport you want to check. The `Unit` parameter of `OpenDevice()` contains either 0 for gameport 1 or 1 for gameport 2:

```
#define GP_LEN (ULONG) (sizeof(struct IOStdReq))
struct IOStdReq   *GamePortRequest = 0L;
...
   Open_A_Device("gameport.device", 1L, &GamePortRequest, 0L,
GP_LEN);
   ...                          /* 1L: GamePort 2 */
   Close_A_Device(GamePortRequest);
```

This sequence opens gameport 2 for access. If you want to use both gameports for a game, you must open the gameport device twice, once with `GamePortRequest = 0L` and with `GamePortRequest = 1L`.

Controller specification

Once `OpenDevice` executes, you must specify the type of controller in the gameport. You can select from the following controller types:

GPCT_MOUSE	(1)	mouse
GPCT_RELJOYSTICK	(2)	relative joystick
GPCT_ABSJOYSTICK	(3)	absolute joystick

If you select `GPCT_MOUSE`, the system accesses the Amiga mouse in gameport 2. Do not try accessing the mouse in gameport 1 through the input device, because the input device treats the mouse in gameport 1 as a different device device. You can, however, instruct the input device to check a joystick instead of a mouse (see Section 4.8—The input device).

Gameport 1 access

The input device controls gameport 1 until you take over the entire system through software, or disable the input task. Avoid disabling the input task, since this can cause problems throughout the system. When you want to use gameport 1 without disturbing the input task, you can access it direct over hardware register `joy0dat` ($dff00a).

The `GP_SETCTYPE` command helps you determine which controller should be checked with `OpenDevice()`. You only need to give the following routine the flags `GPCT_MOUSE`, `GPCT_RELJOYSTICK`, and `GPCT_ABSJOYSTICK`:

```
/****************************************************************
*                        GamePort_SetCType()      (Game_Support)*
*                                                               *
* Function: Determine controller type                           *
*---------------------------------------------------------------*
* Input - Parameter:                                            *
*                                                               *
* GamePortRequest:   GamePort-Device-Block                      *
* Type:              Controller-Type                            *
****************************************************************/

VOID GamePort_SetCType (GamePortRequest,Type)
struct IOStdReq        *GamePortRequest;
UBYTE                                  Type;
{
   UBYTE GP_Type;

   GP_Type                     = Type;
   GamePortRequest->io_Data    = (APTR) &GP_Type;
   GamePortRequest->io_Length = 1l;
   Do_Command (GamePortRequest,(UWORD)GPD_SETCTYPE);
}
```

The address of the variable is given to the gameport device block, which contains the type to be set. Because this type must be established as a UBYTE variable, it contains the value 1L.

Operating systems and controllers

The Amiga's operating system is an ongoing system. Like many software/hardware bases, it has undergone many upgrades and will probably continue to develop. Later versions of Kickstart may support other gameport controllers (e.g., a lightpen). Since this new controller is not supported from the previous Kickstart versions, the program will probably crash if started from an earlier Kickstart version.

To prevent this, the value -1 is returned for `GPDERR_SETCTYPE` for a controller not supported in the `io_Error` of the gameport device block. This can be altered to abort the program without a crash. The controllers `mouse`, `Reljoystick`, and `Absjoystick` are all supported by all versions of Kickstart, so the controller check is unnecessary for these controllers. The following routine helps you determine which controllers are connected to the gameport:

```
/****************************************************************
*                        GamePort_AskCType()      (Game_Support)*
*                                                               *
* Function: Which controller is supported?                      *
*---------------------------------------------------------------*
* Input - Parameter:                                            *
*                                                               *
* GamePortRequest: GamePort-Device-Block                        *
* Type:            Address of the bytes, assigned to the        *
*                  controller type                              *
****************************************************************/

VOID GamePort_AskCType (GamePortRequest,Type)
struct IOStdReq        *GamePortRequest;
UBYTE                                *Type;
{

   GamePortRequest->io_Data   = (APTR) Type;
   GamePortRequest->io_Length = 1l;
   Do_Command (GamePortRequest,(UWORD)GPD_ASKCTYPE);
}
```

Other than the gameport device block, you need the address of the bytes into which the controller type should be placed. If the value zero (`GPCT_NOCONTROLLER`) is returned in `Type`, you can then use the gameport for your own controller. The gameport is occupied by another program when `Type == -1 (GPCT_ALLOCATED)`. When the value 1, 2 or 3 is returned in `Type`, that means that your program is being used by the gameport, and the controller that was returned is supported by the gameport device.

4.7.2 Reading gameport device status

The `READEVENT` command allows the user to read the gameport device status.

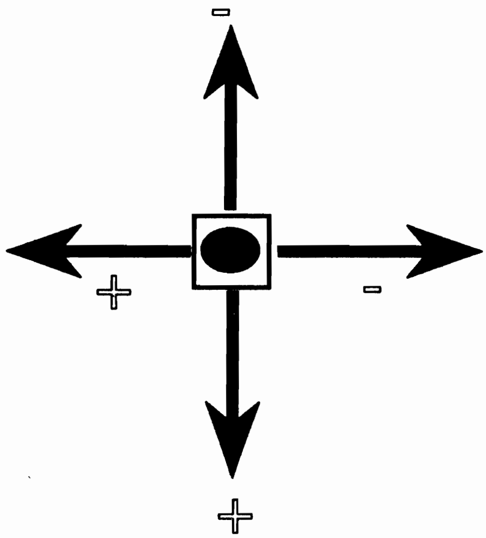
```
/****************************************************************
*                        GamePort_ReadEvent()     (Game_Support)*
*                                                               *
* Function: Read controller status                              *
*---------------------------------------------------------------*
* Input - Parameter:                                            *
*                                                               *
* GamePortRequest: GamePort-Device-Block                        *
* ReadEvent:       Addesse of an InputEvent strukture, in which*
*                  the controller status is written             *
****************************************************************/
```

```
VOID GamePort_ReadEvent (GamePortRequest,ReadEvent)
struct IOStdReq         *GamePortRequest;
struct InputEvent                       *ReadEvent;
{
   GamePortRequest->io_Data    = (APTR) ReadEvent;
   GamePortRequest->io_Length  = (ULONG)(sizeof (struct
InputEvent));
   GamePortRequest->io_Command = (UWORD)GPD_READEVENT;
   DoIO   (GamePortRequest);
}
```

The controller status (e.g., left or right on a joystick) is placed in an input event structure assigned by you (see Section 4.8). `ReadEvent.ie_position.ie_x` and `ReadEvent.ie_position.ie_y` return the controller settings. That differentiates between the three different controller types.

GPCT_MOUSE:

When you read the mouse (or a trackball), the two variables listed above receive the number of steps that the mouse has moved. The faster you move the mouse, the larger the value. When you move the mouse up, the Y coordinate is negative. When you move the mouse down, the Y coordinate is positive. When you move the mouse left, the X coordinate is negative, and when you mouse the mouse right, the X coordinate is positive.



Remember that the value given here represents the movement, rather than the actual position of the mouse pointer. If you want to control one of your own mouse pointers, you only need to add the values in ReadEvent.ie_X and ReadEvent.ie_Y (see Section 4.8) to the position of the mouse pointer.

GPCT_RELJOYSTICK:

When you plug a relative joystick in the gameport, the system returns the stick status in the form of the numbers -1, 0 and 1. The pattern given in the figure above represents the negative and positive values. The center position of the joystick returns a zero to the variables.

GPCT_ABSJOYSTICK:

Movement patterns for the absolute and relative joysticks are the same. The difference between them is that GPCT_RELJOYSTICK constantly supplies the position, while GPCT_ABSJOYSTICK only states when the joystick is brought to a new position.

Mouse and fire buttons

The ReadEvent structure also comes into play for reading the mouse buttons on a mouse or the fire button on a joystick. The ie_Code field describes the status of these buttons. When this element has the value IECODE_LBUTTON (0x68), either the fire button or the left mouse key was pressed. The right mouse key is tested through IECODE_RBUTTON (0x69). You also have the option of reading the release of the button by instructing the gameport device through SETTRIGGER:

```
/****************************************************************
*                    GamePort_SetTrigger()     (Game_Support)*
*                                                              *
* Function: Set movement trigger                               *
*--------------------------------------------------------------*
* Input - Parameter:                                           *
*                                                              *
* GamePortRequest:  GamePort-Device-Block                      *
* GPT:              GamePortTrigger-Structure, which determines*
*                   when the movement message mshould be sent  *
*                   and if the keypress or release of the key  *
*                   should be announced                        *
****************************************************************/

VOID GamePort_SetTrigger (GamePortRequest,GPT)
struct IOStdReq          *GamePortRequest;
struct GamePortTrigger                   *GPT;
{
   GamePortRequest->io_Data   = (APTR) GPT;
   GamePortRequest->io_Length = (ULONG)(sizeof (struct
GamePortTrigger));
   Do_Command (GamePortRequest,(UWORD)GPD_SETTRIGGER);
}
```

Don't let the comments in this routine confuse you: the factors in this function primarily control mouse movement (more on this later).

Keypresses

The given GamePortTrigger structure helps you determine whether one key was pressed (GPTF_DOWNKEYS = 0x01), one key was released (GPTF_UPKEYS = 0x02) or both keys were released (GPTF_UPKEYS+GPTF_DOWNKEYS = 0x03). This determination occurs through the Keys element.

This Keys element is not the only element of the GamePortTrigger structure:

```
Offset   Structure
------   --------          struct GamePortTrigger
         {/* defined in "devices/gameport.h" */
0  0x00     UWORD gpt_Keys;
2  0x02     UWORD gpt_TimeOut;
4  0x04     UWORD gpt_XDelta;
6  0x06     UWORD gpt_YDelta;
8  0x08  }
```

Timeout states the number of vertical blanks that should be sent from the gameport device between ReadEvents (60 per second). XDelta and YDelta state after how many pulses of the controller a ReadEvent should be created. This only works when using a mouse as an input device. The mouse contains two shafts fitted with wheels, and the wheels have holes drilled in them. When the mouse changes position, the holes interrupt a light source. The gameport checks for this light change.

These pulses are added together. When the sum of the values exceeds the values given in XDelta and YDelta, the mouse pointer moves to point XDelta and YDelta. When you give the value 1 for XDelta and YDelta, the mouse pointer moves the fastest. This means that you need less room on your desk to move the pointer from the upper left corner of the screen to the lower right corner of the screen.

Preferences

Preferences program also allows the user to set the mouse speed. The mouse speed assigned after every boot operation is an input device routine which accesses the gameport device.

How does Preferences know which values to set for the mouse speed? GamePort_AskTrigger reads the GamePortTrigger of the corresponding gameport:

```
/*************************************************************
*                    GamePort_AskTrigger()      (Game_Support)*
*                                                             *
* Function: Which condition makes sence for the controller    *
*-------------------------------------------------------------*
* Input - Parameter:                                          *
*                                                             *
* GamePortRequest: GamePort-Device-Block                      *
* GPT:             Address of the GamePortTrigger structure   *
```

```
*                  to be filled  (see SetTrigger)              *
****************************************************************/

VOID GamePort_AskTrigger (GamePortRequest,GPT)
struct IOStdReq          *GamePortRequest;
struct GamePortTrigger                 *GPT;
{

   GamePortRequest->io_Data   = (APTR) GPT;
   GamePortRequest->io_Length = (ULONG)(sizeof (struct
GamePortTrigger));
   Do_Command (GamePortRequest,(UWORD)GPD_ASKTRIGGER);
}
```

This routine fills the `GamePortTrigger` function that was given by you with the trigger values of the corresponding gameport, which you can then read.

4.7.3 A gameport device application

The following program checks a mouse in gameport 2. Left mouse button key presses are displayed; pressing the right mouse button ends the program. When you move the mouse, the position change is displayed on the screen. The faster you move the mouse, the larger the displayed value. Combine the game support routines to make the Game_Support.c file. Include the devs/gameport.h and devs/inputevent.h files in the Game_Suppport.c file.

```
/****************************************************************
*                         GamePort.c                           *
*                     (c) Bruno Jennrich                       *
*                         August 1988                          *
****************************************************************/

/****************************************************************
* Compile-Info:                                                *
*                                                              *
* cc GamePort                                                  *
* ln GamePort.o Game_Support.o Devs_Support.o -lc              *
****************************************************************/

#include "exec/Types.h"
#include "exec/memory.h"
#include "exec/io.h"
#include "exec/devices.h"
#include "devices/inputevent.h"
#include "devices/gameport.h"

#define GP_LEN (ULONG) (sizeof (struct IOStdReq))

struct IOStdReq        *GamePortRequest=0;
struct GamePortTrigger GPT;
struct InputEvent       ReadEvent;
```

```
/*************************************************************
*                        CloseIt()                    (User)*
* Function: In case of an error, release structure and memory *
*------------------------------------------------------------*
* Input - Parameter:                                         *
* String: Error-Message                                      *
*************************************************************/
VOID CloseIt(String)
UBYTE        *String;
{
   UWORD *dff180 = (UWORD *)0xdff180;
   UWORD i;
   UWORD Error;
   Error = 0;
   if (strlen (String) > 0)
   {
      for (i=0;i<0xffff;i++) *dff180 = i;
      Error = 10;
   }
   puts (String);

   if (GamePortRequest != 0) Close_A_Device (GamePortRequest);
   exit (Error);
}
/*************************************************************
*                     The_GamePort_Device()           (User)*
*                                                            *
* Function: Use GamePort device                              *
*************************************************************/
The_GamePort_Device()
{
   Open_A_Device
("gameport.device",1l,&GamePortRequest,0l,GP_LEN);
   printf (" Please put mouse in second GamePort!\n");
   printf (" right mouse key == end of program\n");
   GamePort_SetCType (GamePortRequest,(UBYTE)GPCT_MOUSE);
   GPT.gpt_Keys     = (UWORD)(GPTF_UPKEYS+GPTF_DOWNKEYS);
  /* Announce both */
   GPT.gpt_Timeout = (UWORD) 0;
   GPT.gpt_XDelta  = (UWORD) 1;
   GPT.gpt_YDelta  = (UWORD) 1;

   GamePort_SetTrigger (GamePortRequest,&GPT);
   ReadEvent.ie_Code = 0;
   while (ReadEvent.ie_Code != IECODE_RBUTTON)
   {
      if (ReadEvent.ie_Code == IECODE_LBUTTON) printf ("Left
Button\n");
      if (ReadEvent.ie_Code == IECODE_LBUTTON+IECODE_UP_PREFIX)
printf ("Left Button released\n");
      GamePort_ReadEvent (GamePortRequest,&ReadEvent);
      printf ("x: %d\n y: %d\n",ReadEvent.ie_X,ReadEvent.ie_Y);
      Do_Command (GamePortRequest,(UWORD)CMD_CLEAR);
   }
   GamePort_SetCType (GamePortRequest,(UBYTE)GPCT_NOCONTROLLER);
   Close_A_Device (GamePortRequest);
}

main()
{
   The_GamePort_Device();
}
```

4.8 The input device

The input device is based to some degree on the keyboard device and the gameport device (see Sections 4.6 and 4.7). The input device supports the following commands:

IND_ADDHANDLER	(9)	insert own input handler
IND_REMHANDLER	(10)	remove own handler
IND_WRITEEVENT	(11)	send input event to all other input device users
IND_SETTHRESH	(12)	set time trigger for repeat function
IND_SETPERIOD	(13)	determine repeat speed
IND_SETMPORT	(14)	determine mouse port
IND_SETMTYPE	(15)	determine mouse port controller
IND_SETMTRIG	(16)	determine mouse port controller trigger

These commands are defined in "devices/input.h". In addition to the device commands, the input device supports the following common commands:

CMD_RESET	(1)	reset input device without disabling handler
CMD_CLEAR	(5)	clear input buffer: suppress all previously sent input events and those not yet processed by the handler
CMD_STOP	(6)	stop input device
CMD_START	(7)	start input device

4.8.1 Opening and closing the input device

The input device must be opened before access, using the Open_A_Device() function:

```
#define INPUT_LEN (ULONG) (sizeof(struct IOStdReq))
struct IOStdReq *InputRequest;
...
   Open_A_Device("input.device", 0L, &InputRequest, 0L,
INPUT_LEN);
```

After access the device must be closed again, using the `Close_A_Device()` function, as usual:

```
Close_A_Device (InputRequest);
```

4.8.2 Accessing the input device

No READ command exists for the input device. But how can we receive key presses from the input device and process them? The key phrase here is *input handler*. The input task calls an input handler which controls all of the keypresses and mouse movements. This handler receives an input event structure that was created from the input task. The input event structure looks like this:

```
Offset:     Structure:
-------     ---------
            struct InputEvent
            {/* defined in "devices/inputevent.h" */
0  0x00        struct InputEvent *ie_NextEvent;
4  0x04        UBYTE              ie_Class;
5  0x05        UBYTE              ie_SubClass;
6  0x06        UWORD              ie_Code;
8  0x08        UWORD              ie_Qualifier;
               union
               {
                    struct
                    {
10 0x0A                WORD       ie_x;
12 0x0C                WORD       ie_y;
                    } ie_xy;
10 0x0A             APTR          ie_addr;
               } ie_position;
14 0x0E        struct timeval     ie_TimeStamp;
22 0x16     }
```

Let's take a closer look at this structure. The variable `ie_Class` contains the result type, which the input event structure contains. The following input event classes exist:

IECLASS_NULL	(0x00)	NOP input event
IECLASS_RAWKEY	(0x01)	keyboard code
IECLASS_RAWMOUSE	(0x02)	mouse movement
IECLASS_EVENT	(0x03)	internal event
IECLASS_POINTERPOS	(0x04)	mouse position
!!!IECLASS	(0x05)	does not exist!
IECLASS_TIMER	(0x06)	timer event
IECLASS_GADGETDOWN	(0x07)	gadget clicked on
IECLASS_GADGETUP	(0x08)	gadget released
IECLASS_REQUEST	(0x09)	requester now displayed

IECLASS_MENULIST	(0x0A)	menu clicked on
IECLASS_CLOSEWINDOW	(0x0B)	window close gadget clicked
IECLASS_SIZEWINDOW	(0x0C)	window resized
IECLASS_REFRESHWINDOW	(0x0D)	window refreshed
IECLASS_NEWPREFS	(0x0E)	new Preferences
IECLASS_DISKREMOVED	(0x0F)	disk removed
IECLASS_DISKINSERTED	(0X10)	disk inserted
IECLASS_ACTIVEWINDOW	(0x11)	window activated
IECLASS_INACTIVEWINDOW	(0x12)	window deactivated

Let's examine each input event and how `ie_Class` interprets them:

IECLASS_Null

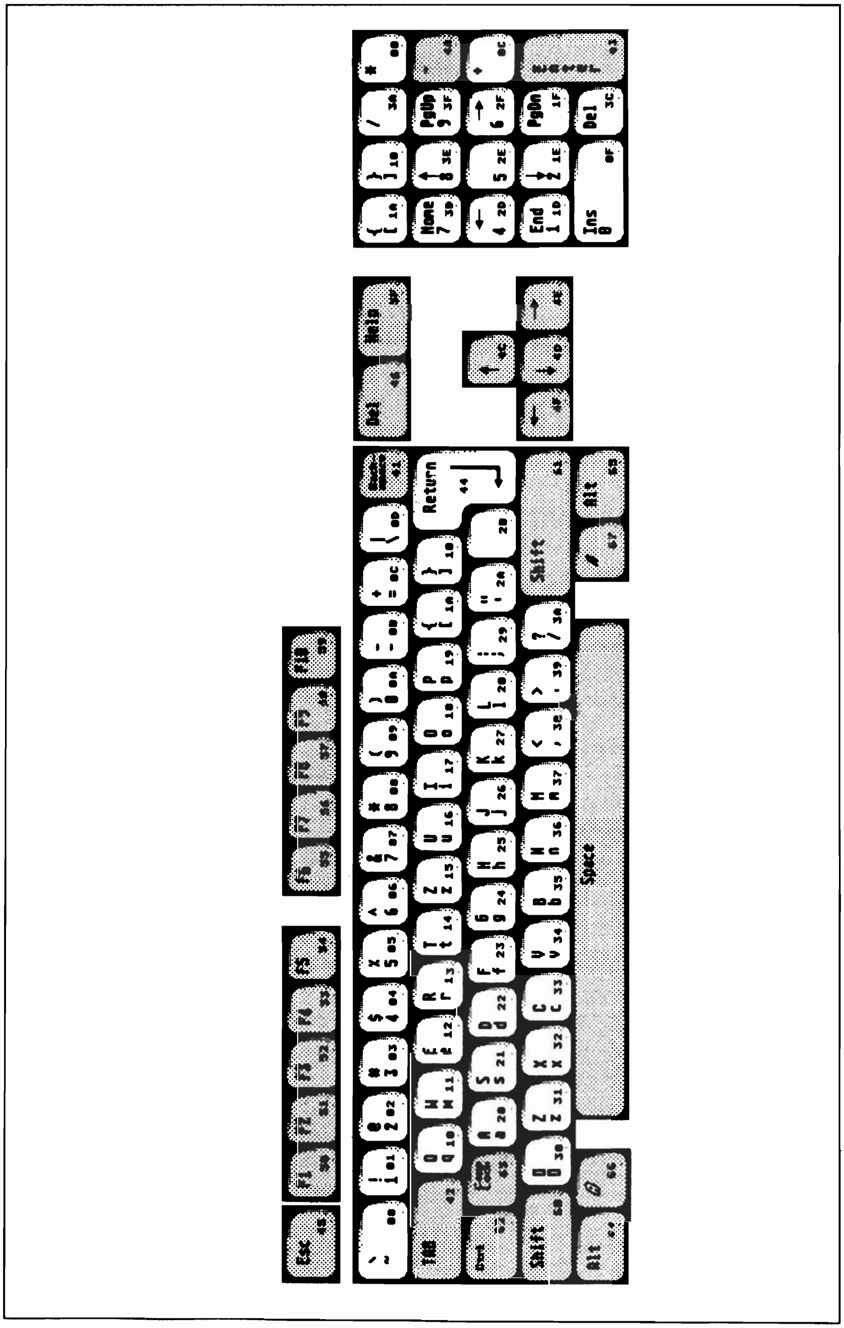
A NOP input event. Data from the event has no meaning.

IECLASS_RAWKEY

Places the keyboard code of the pressed key in `ie_Code` (not the ASCII code). `ie_Qualifier` contains the status of the <Ctrl>, <Shift> and <Alt> keys:

IEQUALIFIER_LSHIFT	(0x0001)	left <Shift> key
IEQUALIFIER_RSHIFT	(0x0002)	right <Shift> key
IEQUALIFIER_CAPSLOCK	(0x0004)	enable <Caps lock>
IEQUALIFIER_CONTROL	(0x0008)	<Ctrl> key
IEQUALIFIER_LALT	(0x0010)	left <Alt> key
IEQUALIFIER_RALT	(0x0020)	right <Alt> key
IEQUALIFIER_LCOMMAND	(0x0040)	left <Amiga> key
IEQUALIFIER_RCOMMAND	(0x0080)	right <Amiga> key
IEQUALIFIER_NUMERICPAD	(0x0100)	numeric keypad key
IEQUALIFIER_REPEAT	(0x0200)	key repeated
IEQUALIFIER_INTERRUPT	(0x0400)	???
IEQUALIFIER_MULTIBROADCAST	(0x0800)	???
IEQUALIFIER_LBUTTON	(0x1000)	left mouse button
IEQUALIFIER_RBUTTON	(0x2000)	right mouse button
IEQUALIFIER_MBUTTON	(0x4000)	middle button (does not exist)
IEQUALIFIER_RELATIVEMOUSE	(0x8000)	relative mouse position message

When the bit is set, the <Shift>, <Alt> or <Ctrl> key was also pressed as well as the key in `ie_Code`. The illustration shows the layout of the key codes. The numbers on the keys are equal to the value returned in `ie_Code`. The two input events are sent from the input device when one key is pressed. The keypress and the release of the key are two separate events. You can differentiate the two input events because the high value bit (bit 7 = 0x80) is set in actual keyboard code (`ie_Code`) when the key is released (IDCMP flag: RAWKEY).



IECLASS_RAWMOUSE

Returns a mouse movement to ie_x and ie_y. These two variables are part of a union. Here are two macros to allow you to move the mouse in the X or Y direction without entering input event ie_position.ie_x time after time:

```
#define ie_X ie_position.ie_x;
#define ie_Y ie_position.ie_y;
```

Now it is possible to access the movement change over input event ie_X to the X coordinate. The IDCMP flag DELTAMOVE must be set in the current window so that the input task of this event can be sent.

IECLASS_EVENT

An internal event of the input device. It is a necessary part of keeping the system informed of changes to the current input window. ie_Code has the value IECODE_NEWACTIVE (0x01).

IECLASS_POINTERPOS

The absolute mouse position as placed in ie_X and ie_Y. This event functions only when the IDCMP flag MOUSEMOVE is set for the current window.

IECLASS_TIMER

Places the current system time in timeval. This event comes from the input task every 60th of a second (IDCMP flag: INTUITICKS).

IECLASS_GADGETDOWN

Places the address of a clicked gadget in ie_Position.ie_addr. This event functions only when the IDCMP flag GADGETDOWN is set for the current window. System gadgets like the front gadget (WINDOW-TO-FRONT) or the back gadget (WINDOW-TO-BACK) cannot be checked.

IECLASS_GADGETUP

Places the address of a released gadget in ie_position.ie_addr. This event functions only when the IDCMP flag GADGETUP is set for the current window.

IECLASS_REQUESTER

Sent when a requester is represented in the current window. IECODE_REQSET (0x01) is in ie_Code with the requester encountered first. When another one is encountered without the first requester disappearing, this event is no longer sent.

When all of the requesters have disappeared, another input event is sent. This time the value IECODE_REQCLEAR (0x00) is given in ie_Code. To receive this input event, the IDCMP flag REQCLEAR and/or REQSET must be set.

IECLASS_MENULIST

Places the code of the menu selected from the current window in ie_Code. This event functions only when the IDCMP flag MENUPICK is set for the current window.

IECLASS_CLOSEWINDOW

Sent when the user clicks on the close gadget of the current window. This event functions only when the IDCMP flag CLOSEWINDOW is set for the current window.

IECLASS_SIZEWINDOW

Sent when the size of the current window changes. This event functions only when the IDCMP flag NEWSIZE is set for the current window.

IECLASS_REFRESHWINDOW

Sent when the current window should be refreshed. This event functions only when the IDCMP flag REFRESHWINDOW is set for the current window.

IECLASS_NEWPREFS

Sent when new preferences are present due to changes from the Preferences program (IDCMP flag: NEWPREFS).

IECLASS_DISKREMOVED

Sent when the user removes a disk from a disk drive (IDCMP flag: DISKREMOVED).

IECLASS_DISKINSERTED

Sent when the user inserts a disk in a disk drive (IDCMP flag: DISKINSERTED).

IECLASS_ACTIVEWINDOW

Makes the window most recently clicked the active window.

IECLASS_INACTIVEWINDOW

Deactivates the currently active window.

Intuition and the input device

Many of the input/output functions used in Intuition are controlled through the input device, and must be passed on to your Intuition window. Now let's write a routine that will let you view the events sent by the input task. We'll start by creating an input handler using the IND_ADDHANDLER device command:

```
ULONG  User_Routine;
VOID   Input_Code();
struct Interrupt Input_Handler;
```

```
/****************************************************************
*                        Input_AddHandler()      (Input_Support)*
*                                                               *
* Function: Add own C handler in input handler                  *
*---------------------------------------------------------------*
* Input - Parameter:                                            *
*                                                               *
* InputRequest: Input-Device-Block                              *
* Handler:      Address of your own handler routine (C)         *
* Data:         Address of data area for handler routine        *
****************************************************************/

VOID Input_AddHandler (InputRequest,Handler,Data)
struct IOStdReq       *InputRequest;
VOID                               *Handler;
APTR                                        Data;
{
   User_Routine                   = (ULONG) Handler;
   Input_Handler.is_Data          = Data;
   Input_Handler.is_Code          = (VOID (*)())Input_Code;
   Input_Handler.is_Node.ln_Pri   = 51;

   InputRequest->io_Data          = (APTR)&Input_Handler;
   Do_Command(InputRequest, (UWORD) IND_ADDHANDLER);
}
```

This routine specifies the device block with which the device was opened (or a copy of the original device block). Then the routine gives the address of the handler routine that was written in C. You also have the option of specifying a range of memory for use by your handler routine (e.g., for variables).

Unfortunately the input task is not in the position to call C routines, because the parameters are passed in the stack in C. The input task gives the parameters (input event and data region) in the hardware registers. For this reason, we must switch to an interface that moves the parameters from the hardware register onto the stack, and then call the C program. We have developed a short machine language routine for this purpose:

```
/****************************************************************
*                        _Input_Code.asm        (Input_Support)*
*                                                               *
* Function: Call Input-Handler Routine (C-Routine)              *
*---------------------------------------------------------------*
* Input - Parameter:                                            *
*                                                               *
* A0: Pointer to Input-Event                                    *
* A1: pointer to your own data                                  *
*---------------------------------------------------------------*
* Return values:                                                *
*                                                               *
* D0: Contains the Input-Event to process further               *
****************************************************************/

#asm
   public _geta4
   public _Input_Code
```

```
_Input_Code:
   move.l   a4,-(sp)
   jsr      _geta4
   movem.l  a0/a1,-(sp) ; parameter auf Stack

   move.l   _User_Routine,a0
   jsr      (a0)

   movem.l  (sp)+,a0/a1
   move.l   (sp)+,a4
   rts
#endasm
```

Because the Aztec compiler accesses program variables through hardware register A4, this register must be re-initialized before each call of the C handler routine. This is done here using the routine geta4, which transfers the variables' basis address according to A4. The initialization of register A4 must occur because register A4 contains another value during the input handler processing. This register should return to its old value when leaving the handler. The parameter can be brought to the stack after initialization of register A4, and the C routine can be recalled.

This machine language routine is given as the actual input handler in Input_AddHandler(). For this an interrupt structure is stored whose is_Code field is loaded with the address of the machine language routine, and whose is_Data field is loaded with the address of the memory range. We set the priority of our handler higher than that of the system input handler, so that we get all of the input events that the input task creates first. The priority of the system input handler is 50, and the priority of our handler is 51.

Then the initialized interrupt structure can be given in our input device block (io_Data), and the IND_ADDHANDLER command is sent. Remember not to use any variables named Input_Handler and User_Routine, just in case you want to use the above program. InputHandler is the interrupt structure, over which the machine language segment Input_Code is called. User_Routine contains the address of the C routine that was called, and which the input handler should process. When you want to exit a program that has installed an input handler, you must remove the input handler again. This is done with the following code.

```
/****************************************************************
*                    Input_RemHandler()        (Input_Support)*
*                                                              *
* Function: Disable Input-Handler                              *
*--------------------------------------------------------------*
* Input - Parameter:                                           *
*                                                              *
* InputRequest: Input-Device-Block                             *
****************************************************************/
```

```
VOID Input_RemHandler (InputRequest)
struct IOStdReq       *InputRequest;
{
   InputRequest->io_Data    = (APTR) &Input_Handler;
   Do_Command (InputRequest,(UWORD)IND_REMHANDLER);
}
```

These two routines give you the information needed to write a macro recorder that registers all of the encountered input events. Here's the code for just such a recorder. Combine all the input support routines presented in this section to make the Input_Support.c file. The header for the Input_Support.c file is presented at the end of this section.

```
/*****************************************************************
*                          Recorder.c                            *
*                          August 1988                           *
*                      (c) Bruno Jennrich                        *
*****************************************************************/
/*****************************************************************
*  Compile-Info:                                                 *
*  cc Recorder                                                   *
*  ln Recorder.o Input_Support.o Devs_iSupport.o -lc             *
************** **************************************************/
#include "exec/types.h"
#include "exec/memory.h"
#include "exec/interrupts.h"
#include "exec/nodes.h"
#include "devices/input.h"
#include "devices/inputevent.h"
VOID *AllocMem();
VOID *Open();
ULONG User_Routine;
#define MODE_NEWFILE    1006L
#define INPUT_LEN       (ULONG) (sizeof (struct IOStdReq))
#define RECORD_SIZE     50001
#define INEV_LEN        ((ULONG) (sizeof (struct InputEvent)))
#define MEMTYPE         (MEMF_CHIP | MEMF_CLEAR)
ULONG HowMuchEvents;
ULONG ActualEvents;
BOOL  End;
UWORD *FileHandle=0l;
struct InputEvent *Recorder=0l;
struct IOStdReq   *InputRequest = 0l;
struct InputEvent *Pointer;
/*****************************************************************
*                              CloseIt()                   (User)*
* Function:   Release all occupied memory and structures         *
*----------------------------------------------------------------*
* Input - Parameter:                                             *
* String: Error-Message                                          *
*****************************************************************/
CloseIt (String)
BYTE    *String;
{
   UWORD *dff180 = (UWORD *)0xdff180;
   UWORD i;
   UWORD Error;
   Error = 0;
   if (strlen (String) > 0)
   {
```

```
      for (i=0;i<0xffff;i++) *dff180 = i;
      Error = 10;
   }
   puts (String);
   if (FileHandle != 0)    Close (FileHandle);
   if (InputRequest != 0l) Close_A_Device (InputRequest);
   if (Recorder != 0)      FreeMem
(Recorder,HowMuchEvents*INEV_LEN);
   exit (Error);
}
/****************************************************************
*                         C_Handler()                     (User)*
*                                                               *
* Function: Handles InputEvents of the Input-Task               *
*---------------------------------------------------------------*
* Input - Parameter:                                            *
* Input: InputEvent                                             *
* Data:  Pointer to data                                        *
*---------------------------------------------------------------*
* Return value:                                                 *
* InputEvent, that should be processed further.                 *
****************************************************************/
struct InputEvent *C_Handler(Input,Data)
struct InputEvent           *Input;
ULONG                             *Data;
{
   if (!End)
   {
   if (Input->ie_Class == IECLASS_RAWKEY)
   {
      if (Input->ie_Code == 0x45) End = TRUE;    /* Escape */
      else
      if (*Data < HowMuchEvents)
      {
         Pointer          = &Recorder[*Data];  /* InputEvent  */
         Pointer->ie_NextEvent          = 0;
         Pointer->ie_Class              = Input->ie_Class;
         Pointer->ie_SubClass           = Input->ie_SubClass;
         Pointer->ie_Code               = Input->ie_Code;
         Pointer->ie_Qualifier          = Input->ie_Qualifier;
         Pointer->ie_TimeStamp.tv_secs  = Input-
>ie_TimeStamp.tv_secs;
         Pointer->ie_TimeStamp.tv_micro = Input-
>ie_TimeStamp.tv_micro;
         *Data+=1;
      }
      else End = TRUE;
   }
   }
   return (Input);
}
/****************************************************************
*                     Input_Device()                      (User)*
*                                                               *
* Function: Use InputDevice                                     *
****************************************************************/
Input_Device()
{
   Open_A_Device ("input.device",0l,&InputRequest,0l,INPUT_LEN);

   End = FALSE;
   ActualEvents = 0;
```

```
   Input_AddHandler (InputRequest,C_Handler,&ActualEvents);

   while (!End);            /* wait until end of the line */

   Input_RemHandler (InputRequest);
   Close_A_Device   (InputRequest);
}
/****************************************************************
*                          main()                          (User)*
*                                                               *
* Input - Parameter:                                            *
* argv[1]: Name of the Macro File                               *
* argv[2]: Number of events                                     *
****************************************************************/
main (argc,argv)
UWORD argc;
BYTE     **argv;
{
   if (argc != 3)
   {
      printf ("USAGE: %s MacroFile How many events\n",argv[0]);
      CloseIt ("");
   }

   if ((HowMuchEvents = atoi(argv[2])) < 0)/* how many events */
      CloseIt ("HowMuchEvents < 0 !!!");

   Recorder = (struct InputEvent *) AllocMem
(HowMuchEvents*INEV_LEN,MEMTYPE);
                                     /* Get memory for events  */
   if (Recorder == 0l)
      CloseIt ("No Memory for InputEvents !!!");

   Input_Device();

   FileHandle = Open (argv[1],MODE_NEWFILE);

   if (FileHandle == 0l)
      CloseIt ("Cannot Open MacroFile !!!");

   Write (FileHandle,&ActualEvents,4l);        /* save length */
   Write (FileHandle,Recorder,ActualEvents*INEV_LEN);
                                       /* save events */
   Close (FileHandle);

   FreeMem (Recorder,HowMuchEvents*INEV_LEN);
}
```

Program description

This program first opens the input device. Then the input handler is installed. This handler calls our `C_Handler` routine. Only RAWKEY events are recorded in this `C_Handler` routine. Whether it was recorded or not, the received input event is given in the other input handler (all of the input handlers like interrupt server are organized in a list). This happens by simply returning the received input event. The `return()` routine writes the address of the received input event in D0 for this. If you place the value 0 in D0, you must remember that the following handler can no longer be accessed.

Also, remember that input events can be combined. The `ie_NextEvent` array uses one of each input event structures (this pointer points to the input event's successor). So it can happen that your input handler only receives the first input event of a long list. You are free to change this list to insert your own input events. Just bear in mind that anyone developing an input event handler is responsible for any problems he/she creates.

Now back to the above program. We have recorded only RAWKEY events there. With this you can record keypresses that occur while the program is running (it doesn't matter in which window, as long as no input handler with a higher priority exists). For this you must give the maximum number of events that should be recorded. You must also give the names of files in which these recorded input events should be stored. A sample call of the program can look like this:

```
Recorder Macro 100
```

This program call allows you to save up to 50 keypresses to a file named `Macro`. You can only record up to 50 keypresses because a keypress consists of depressing and releasing the key. An input event is sent for each, so one keypress actually counts for two actions. But you can test bit 7 in `ie_Code` and if this is clear, the corresponding event is not recorded.

Pressing the <Esc> key aborts the program—all data up to that point is saved and the program ends. A macro recorder is worthwhile only if you can play back the recorded key presses. We can also play back the input task with the help of this function. We send only input events in the input handler, from which more can be given in the system, for example the system handler and our own input handler installed with `Input_AddHandler`. This function is called `WRITEVENT`. Add it to your Input_support.c file, remember to include devices/inputevent.h before compiling.

```
#define INEV_LEN        ((ULONG) (sizeof(struct InputEvent)))
/*************************************************************
*                  Input_WriteEvent()           (Input_Support)*
*                                                            *
* Function: Givw InputEvent in other Input-Handler again      *
*------------------------------------------------------------*
* Input - Parameter:                                         *
*                                                            *
* InputRequest: Input-Device-Block                           *
* Event:        InputEvent to redirect                       *
*************************************************************/

VOID Input_WriteEvent (InputRequest,Event)
struct IOStdReq       *InputRequest;
struct InputEvent                  *Event;
{
   InputRequest->io_Data     = (APTR) Event;
```

```
   InputRequest->io_Length  = INEV_LEN;
   InputRequest->io_Flags   = (UBYTE)0;

   Do_Command (InputRequest,(UWORD)IND_WRITEEVENT);
}
```

This routine lets you send an input event to all of the present handlers. If you installed your own handler, this handler also receives your input event, as long as a handler with a higher priority has not changed the input events.

Based on the above routine we can actually write a program to play back the entered keypresses. Our program must open the file created by the recorder, read the saved number of recorded input events, allocate memory as needed and load the input events into this allocated memory. Then these input events only need to be executed by `Input_WriteEvent()`:

```
/****************************************************************
*                          Play.c                               *
*                        August 1988                            *
*                    (c) Bruno Jennrich                         *
****************************************************************/
/****************************************************************
* Compile-Info:                                                 *
*                                                               *
* cc Play                                                       *
* ln Play.o Input_Support.o Devs_Support.o -lc                  *
****************************************************************/
#include "exec/types.h"
#include "exec/memory.h"
#include "exec/interrupts.h"
#include "exec/nodes.h"
#include "devices/input.h"
#include "devices/inputevent.h"

VOID *AllocMem();
VOID *Open();
#define MODE_OLDFILE    1005L
#define INPUT_LEN       (ULONG) (sizeof (struct IOStdReq))
#define RECORD_SIZE     50001
#define INEV_LEN        ((ULONG) (sizeof (struct InputEvent)))
#define MEMTYPE         (MEMF_CHIP | MEMF_CLEAR)
UWORD              *FileHandle   = 01;
struct InputEvent  *Player       = 01;/* Memory for InputEvents */
ULONG               Length       = 01;/* Niumber of InputEvents */
struct IOStdReq    *InputRequest= 01;

/****************************************************************
*                        CloseIt()                        (User)*
*                                                               *
* Funktion: In case of erroe, release memory and structures     *
*---------------------------------------------------------------*
* Input - Parameter:                                            *
*                                                               *
* String: Error-String                                          *
****************************************************************/
VOID CloseIt (String)
```

```
BYTE        *String;
{
   UWORD *dff180 = (UWORD *)0xdff180;
   UWORD i;
   UWORD Error;

   Error = 0;

   if (strlen (String) > 0)
   {
      for (i=0;i<0xffff;i++) *dff180 = i;
      Error = 10;
   }

   puts (String);

   if (FileHandle != 0l)    Close (FileHandle);
   if (Player != 0l)        FreeMem (Player,Length*INEV_LEN);
   if (InputRequest != 0l)  Close_A_Device (InputRequest);

   exit (Error);
}
/***************************************************************
*                          main                         (User)*
* Input - Parameter:                                          *
* argv[1]:   MacroFile                                        *
***************************************************************/
main (argc,argv)
UWORD argc;
BYTE     **argv;
{
   UWORD i;
   if (argc != 2)
   {
      printf ("USAGE: %s MacroFile\n",argv[0]);
      CloseIt ("");
   }
   FileHandle = Open (argv[1],MODE_OLDFILE);
   if (FileHandle == 0) CloseIt ("File does not exist!!!");
   if (Read (FileHandle,&Length,4l) != 4)/* Number of events  */
      CloseIt ("Read Error #1 !!!");      /* Events cannot be  */
                                          /* read              */
   Player = AllocMem (Length*INEV_LEN,MEMTYPE);
   if (Player == 0) CloseIt ("No Memory !!!");
                        /* Memory for Length-Events occupied*/

   if (Read (FileHandle,Player,Length*INEV_LEN) !=
Length*INEV_LEN)
      CloseIt ("Read-Error #2 !!!");

   Close (FileHandle);

   Open_A_Device ("input.device",0l,&InputRequest,0l,INPUT_LEN);

   for (i=0;i<Length;i++)     /* rather small cost, isn't it? */
      Input_WriteEvent (InputRequest,&Player[i]);

   Close_A_Device (InputRequest);

   FreeMem (Player,Length*INEV_LEN);
}
```

This program shows you how to open and use a device with as little effort as possible. The program section that sends out the input events consist of only three functions and a control instruction (`for(;;);`). You find the compiled programs Recorder and Play on the optional disk in the `CH-4/4.8` directory. We have also included a macro in this directory. You can play it back by entering the directory and entering:

```
Play Macro
```

4.8.3 The input device, mouse and keyboard

Now let's look at the input device commands which access the mouse and keyboard. You may want to re-read Section 4.7, since much of the material concerning the gameport device also applies to the input device.

This section also examines the similarities between the Preferences program and the input device.

Setting repeat speed

The Preferences program offers the option of changing the key repeat speed. This parameter controls the speed at which a key repeats when the user presses and holds the key. The key repeat speed is set from Preferences using slider gadgets. When booting, the system reads the repeat rate from the `devs/system` configuration file and places the value in `IND_SETTHRESH`. If you want to change the repeat speed after booting, you must load the Preferences program, change the repeat speed and exit to the Workbench. The following routine allows you to change the repeat speed without returning to Preferences:

```
/****************************************************************
*                        Input_SetPeriod()      (Input_Support)*
*                                                               *
* Function: Set repeat period                                   *
*---------------------------------------------------------------*
* Input - Parameter:                                            *
*                                                               *
* InputRequest: Input-Device-Block                              *
* Secs, Micro:  Time that should pass between two repetitions  *
****************************************************************/

VOID Input_SetPeriod     (InputRequest,Secs,Micro)
struct timerequest       *InputRequest;
ULONG                                   Secs,Micro;
{
   InputRequest->tr_time.tv_secs  = Secs;
   InputRequest->tr_time.tv_micro = Micro;

   Do_Command (InputRequest,(UWORD)IND_SETPERIOD);
}
```

You only need to give this routine the time that should pass between two repetitions of a pressed key.

Changing the repeat threshold

When you press and hold a key, the key delays before beginning key repeat. The time that passes between the pressing down and the repetition of a key is called the *repeat threshold*. You can completely change this threshold with the following routine.

```
/****************************************************************
*                    Input_SetThresh()       (Input_Support)*
*                                                              *
* Function: Set repeat time thresholden                        *
*--------------------------------------------------------------*
* Input - Parameter:                                           *
*                                                              *
* InputRequest: Input-Device-Block                             *
* Secs, Micro:  After how many seconds and micro seconds the   *
*               key should be repeated                         *
****************************************************************/

VOID Input_SetThresh    (InputRequest,Secs,Micro)
struct timerequest      *InputRequest;              /* !!!! */
ULONG                                 Secs,Micro;
{
   InputRequest->tr_time.tv_secs  = Secs;
   InputRequest->tr_time.tv_micro = Micro;

   Do_Command (InputRequest,(UWORD)IND_SETTHRESH);
}
```

Try a threshold of zero seconds and zero microseconds. Run the program, press a key and watch the result.

Setting the mouse speed

Now from the keyboard to the mouse. Here's one of the input device functions usually reached through Preferences:

```
/****************************************************************
*                    Input_SetMTrig()        (Input_Support)*
* Function: Set threshold for mouse movement                   *
*--------------------------------------------------------------*
* Input - Parameter:                                           *
* InputRequest:  Input-Device-Block                            *
* Keys:          Mouse button pressed or released?             *
* Timeout:       Send mouse report after how many VBlanks?     *
* XDelta,YDelta: Announce after n mouse movements?             *
****************************************************************/
VOID Input_SetMTrig  (InputRequest,Keys,Timeout,XDelta,YDelta)
struct IOStdReq      *InputRequest;
ULONG                          Keys,Timeout,XDelta,YDelta;
{
   struct GamePortTrigger GPT;
   GPT.gpt_Keys    = Keys;
   GPT.gpt_Timeout = Timeout;
   GPT.gpt_XDelta  = XDelta;
   GPT.gpt_YDelta  = YDelta;
   InputRequest->io_Data        = (APTR) &GPT;
   InputRequest->io_Length      = (ULONG) (sizeof (struct
GamePortTrigger));
```

```
   Do_Command (InputRequest,(UWORD)IND_SETMTRIG);
}
```

This routine informs the input device of how many pulses the mouse should announce for one move in the X or Y direction. As mentioned above, there are two shafts fitted with wheels perpendicular to each other inside of the mouse. These wheels have holes drilled in them. When moving the mouse, each wheel rotates, interrupting a light source. This pulse is then sent to the input device, or the input device reads this pulse from the hardware register.

When the number of this pulse of the given value has been reached, the input device ensures that the mouse pointer moves to a point on the screen. `XDelta` and `YDelta` give the number of the pulse in the X and Y direction by which the mouse pointer was moved to a point on the screen. The larger this value is, the slower the mouse is.

The `Timeout` parameter helps you determine the number of vertical blanks after which a mouse report or mouse event should be sent, in case the mouse is not moved extensively. The parameter has the value 1 (for Intuition). The mouse position is renewed after each vertical blank in the Intuition window, or on the Intuition screen. The `Keys` parameter allows the system to announce the release of a mouse button (`GPTF_UPKEYS`) instead of pressing down a mouse button (GPTF_DOWNKEYS).

Assigning the mouse port

The following routine allows the user to connect the mouse to the second gameport instead of the default first gameport.

```
/****************************************************************
*                    Input_SetMPort()         (Input_Support)*
* Function: Set mouse port                                      *
*---------------------------------------------------------------*
* Input - Parameter:                                            *
* InputRequest: Input-Device-Block                              *
* Port:         0: GamePort 1                                   *
*               1: GamePort 2                                   *
****************************************************************/
VOID Input_SetMPort  (InputRequest,Port)
struct IOStdReq      *InputRequest;
UBYTE                          Port;
{
   UBYTE PointerToPort;
   PointerToPort = Port;
   InputRequest->io_Data        = (APTR) &PointerToPort;
   InputRequest->io_Length      = (ULONG) 1;

   Do_Command (InputRequest,(UWORD)IND_SETMPORT);
}
```

You must give this routine the input device block and a one or zero. Then, depending on which value you entered, the mouse is read from gameport 1 or gameport 2. Remember that the address must be given on the value 1 or 0 with this command.

Joystick as mouse

You can change the mouse controller as well as the mouse port. For example, this means that a joystick can take over almost all the functions of the mouse, except for the right mouse button. The following routine executes this.

```
/***************************************************************
*                         Input_SetMType()        (Input_Support)*
* Function: Set mouse controller type                          *
*--------------------------------------------------------------*
* Input - Parameter:                                           *
* InputRequest: Input-Device-Block                             *
* Type:         new controller type                            *
***************************************************************/
VOID Input_SetMType  (InputRequest,Type)
struct IOStdReq      *InputRequest;
UBYTE                              Type;
{
   UBYTE MouseType;
   MouseType = Type;
   InputRequest->io_Data         = (APTR) &MouseType;
   InputRequest->io_Length       = 11;
   Do_Command (InputRequest,(UWORD)IND_SETMTYPE);
}
```

The new controller type is given in `Type`:

```
GPCT_MOUSE:          mouse in the port
GPCT_RELJOYSTICK:    relative joystick in the port
GPCT_ABSJOYSTICK:    absolute joystick in the port
GPCT_NOCONTROLLER:   nothing more in the port
```

Remember that this command must include the address of the type. That is why the given parameter is written in an extra variable, whose address is given in the device block.

The following is the start of the Input_Support.c file:

```
/***************************************************************
*                      Input_Support.c                         *
* Compile-Info:        cc Input_Support                        *
***************************************************************/
#include "exec/types.h"
#include "exec/memory.h"
#include "exec/interrupts.h"
#include "exec/nodes.h"
#include "devices/input.h"
#include "devices/gameport.h"
#include "devices/timer.h"
#include "devices/inputevent.h"
ULONG User_Routine;
#define INEV_LEN        ((ULONG) (sizeof (struct InputEvent)))
VOID Input_Code();
struct Interrupt Input_Handler;
```

4.9 The console device

The console device is the simplest method used to display text on the screen. A window must first be opened. The console device can read your input from this window, and the console device can activate the output. The opened window is specified before the opening of the console device in the corresponding device block.

```
struct IOstReq *ConsoleRead = 0L;
#define CON_LEN ( ULONG) (sizeof(struct IOStdReq))
...
   Window = OpenWindow(&NewWindow);
   ConsoleRead = (struct IOstdReq *)GetDeviceBlock(CON_LEN);
   ConsoleRead->io_Data   = (APTR) Window;
   ConsoleRead->io_Length = (ULONG) (sizeof(struct Window));
   Open_A_Device("console.device", 0L, &ConsoleRead, 0L, 0L);
```

You can now access the console device. The console device supports the following commands:

CMD_READ	(2)	read keypress
CMD_WRITE	(3)	display text
CMD_CLEAR	(5)	clear console buffer
CD_ASKKEYMAP	(9)	send current keyboard layout
CD_SETKEYMAP	(10)	set new keyboard arrangement
CD_ASKDEFAULTKEYMAP	(11)	find out default keyboard arrangement
CD_SETDEFAULTKEYMAP	(12)	set default keyboard arrangement

As you see, the console device uses few commands, but they play an important role in developing such complicated applications as editors (see DiskEd), word processors, etc.

The console device is controlled through command strings. These are sent to the console device with the help of the command `CMD_WRITE`. You should prepare another device block for the `CMD_WRITE` command so that the `READ` and `WRITE` commands do not interfere with each other:

```
struct IOStdReq *ConsoleWrite = 0L;
...
   ConsoleWrite = (struct IOStdReq*) GetDeviceBlock(CON_LEN);
   Console_Copy(ConsoleRead, ConsoleCopy);
...
```

Combine the following routines labled (Con_Support) to form the Con_Support.c file, don't forget to include the proper header files. Some structure elements must be borrowed for accessing the second device block:

```
/****************************************************************
*                       Console_Copy()            (Con_Support)*
*                                                              *
* Function:  Device-Block copy                                 *
*--------------------------------------------------------------*
* Input - Parameter:                                           *
*                                                              *
* OldStdReq: Original                                          *
* NewStdReq: Copy                                              *
****************************************************************/
VOID Console_Copy (OldStdReq, NewStdReq)
struct IOStdReq   *OldStdReq,*NewStdReq;
{
   NewStdReq->io_Device =
      OldStdReq->io_Device;

   NewStdReq->io_Unit =
      OldStdReq->io_Unit;
}
```

The following routine reads keypresses with `Console_Read()` and `Console_Write()` and displays them on the screen:

```
/****************************************************************
*                     Console_Write()             (Con_Support)*
*                                                              *
* Function: Display string on Console-Device                   *
*--------------------------------------------------------------*
* Input - Parameter:                                           *
*                                                              *
* ConWrite: Device-Block                                       *
* String:   String to be displayed                             *
* Len:      Length of string                                   *
****************************************************************/

VOID Console_Write (ConWrite,String,Len)
struct IOStdReq    *ConWrite;
BYTE                        *String;
ULONG                               Len;
{
   ConWrite->io_Data   = (APTR) String;
   ConWrite->io_Length = Len;
   Do_Command (ConWrite,(UWORD)CMD_WRITE);
}

/****************************************************************
*                     Console_Read()              (Con_Support)*
*                                                              *
* Function: Read string from Console-Device                    *
*--------------------------------------------------------------*
* Input - Parameter:                                           *
* ConWrite: Device-Block                                       *
* String:   Address of the buffers                             *
* Len:      How many characters to be read                     *
****************************************************************/
```

```
VOID Console_Read (ConRead,String,Len)
struct IOStdReq   *ConRead;
BYTE                    *String;
ULONG                          Len;
{
   ConRead->io_Data    = (APTR) String;
   ConRead->io_Length  = Len;
   ConRead->io_Command = (UWORD) CMD_READ;

   DoIO(ConRead);
}
```

In addition to the device block, these routines give you the address of the string to be displayed, or the address of the range of memory into which the read keypress should be written. It also gives you the number of characters to be displayed or read. If you enter -11 as the number of characters to be displayed, all of the strings that end with a null byte are displayed.

When you read keypresses from the keyboard by means of `Console_Read()`, it returns the ASCII code that represents the depressed key. Remember that this is not automatically displayed. You must provide for the output yourself by means of `Console_Write()` or `CMD_WRITE`. As was already mentioned, a certain number of control strings exist with which you can test for the form of the input and output:

BELL (0x7)

Screen flashes and tone sounds.

BACKSPACE (0x08)

Moves the cursor one position to the left. The character to the left of the cursor is not deleted. You can delete the character by combining a backspace and a space.

LINE FEED (0x0a)

Moves the cursor one position down. When the cursor arrives at the bottom line of the screen, the screen scrolls up one line (see SET MODE).

VERTICAL TAB (0x0b)

Moves the cursor one line up. When the cursor arrives at the top line of the screen, the screen scrolls down one line.

FORM FEED (0x0c)

Clears the console window.

CR (0x0d)

Places the cursor in the first column but not in the next line.

SHIFT IN (0x0e)

Enables <Shift> key.

SHIFT OUT (0x0f)

Disables <Shift> key. Only capital letters are displayed in the output.

CAPS LOCK Key

See keyboard layout.

ESC (0x1b)

<Escape> key.

CSI (0x9b)

Control Sequence Introducer. All command strings begin with this ASCII code.

RESET ("<CSI>c")

Resets the console device.

INSERT [N] SPACES ("<CSI>[N]@")

Inserts N spaces starting at the current cursor position. When N is omitted, one space is inserted. N is a decimal string. For example, "<CSI>12@" inserts 12 spaces. @ represents the ASCII code 64.

CURSOR UP [N] ("<CSI>[N]A")

Moves the cursor N lines up. When N is omitted, the cursor moves up one line. N is a decimal string. For example, "<CSI>2A" moves the cursor up two lines.

CURSOR DOWN [N] ("<CSI>[N]B")

Moves the cursor N lines down. When N is omitted, the cursor moves down one line. N is a decimal string. For example, "<CSI>2B" moves the cursor down two lines.

CURSOR FORWARD [N] ("<CSI>[N]C")

Moves the cursor N columns to the right. When N is omitted, the cursor moves right one column. N is a decimal string. For example, "<CSI>20C" moves the cursor 20 characters to the right.

CURSOR BACKWARD [N] ("<CSI>[N]D")

Moves the cursor N columns to the left. When N is omitted, the cursor moves left one column. N is a decimal string. For example, "<CSI>20D" moves the cursor 20 characters to the left.

CURSOR NEXT LINE [N] ("<CSI>[N]E")

Moves the cursor N lines down and to the first column (has the same effect as pressing the <Return> key).

CURSOR PRECEDING LINE [N] ("<CSI>[N]F")

Moves the cursor N lines up and to the first column.

MOVE CURSOR ("<CSI>[N][;M]H")

Moves cursor to line N and, if given, column M. If the M parameter is omitted, the semicolon must be omitted as well. For example, "<CSI>1;1H" or "<CSI>1H" or "<CSI>H" each places the cursor in the upper left corner of the screen (same as Cursor home).

ERASE TO END OF DISPLAY ("<CSI>J")

Clears the screen from the current cursor position to the end of the screen. To clear the entire screen, you can use the following sequence: "<CSI>1;1;H" (Cursor home) and "<CSI>J" (Delete).

ERASE TO END OF LINE ("<CSI>K")

Deletes line from the current cursor position to the end of the current line (same as <Ctrl><Y> function in ED).

INSERT LINE ("<CSI>L")

Inserts a line at the current cursor position.

DELETE LINE ("<CSI>M")

Deletes the line at the current cursor position. Lines below the deleted line scroll up to fill in the deletion (same as <Ctrl><B> function in ED).

DELETE CHARACTER ("<CSI>[N]P")

Deletes N characters to the right of the current cursor position. If the N parameter is omitted, only the character at the current cursor position is deleted.

SCROLL UP [N] LINES ("<CSI>[N]S")

Scrolls the entire screen up N lines. The blank lines below fill with blank spaces.

SCROLL DOWN [N] LINES ("<CSI>[N]T")

Scrolls the entire screen down N lines. The blank lines above fill with blank spaces.

SET MODE ("<CSI>20h")

Treats a line feed as <Return><Line feed> (0x0c,0x0a). The cursor moves to the first column of the next line.

RESET MODE ("<CSI>20L")

Treats a line feed as <Line feed> only when a line feed is sent (0x0a), the cursor moves to the next line but remains in the same column.

DEVICE STATUS REPORT ("<CSI>6n")

Instructs the console device to send a status report in the following form: "<CSI>Line;ColumnR". There are decimal strings in Line and Column which give the line and column of the cursor position. This report can be read using CMD_READ. Remember that you must convert the decimal strings to decimal values if you want to calculate them to determine a new position.

SELECT GRAPHIC Style ("<CSI><Style>;<Foreground>; <Background>m")

Selects character attribute (style), foreground color and background color.

<Style> The Style parameter allows the following values:

0	normal text
1	bold print
3	italics
4	underline
7	inverse

You give the foreground and background color of the characters to be displayed with Foreground and Background:

<Foreground>

30	Color 0
31	Color 1
...	
37	Color 7

<Background>

40	Color 0
41	Color 1
...	
47	Color 7

For example, if you want the text displayed underlined and bold and in colors 0 and 1, you must send the following sequence:

```
"<CSI>4;30;40m"        (underline, without color)
"<CSI>1;30;41m"        (bold, with color)
```

All parameters must be entered.

SET PAGE LENGTH ("<CSI><LEN>t")

Specifies the number of lines that the console device should control in the window. The entire window is usually allocated for text output, but you can reduce the text area with this and the following commands.

When changing the size of the window the console device calculates the new value from the current character set and alters the size of the text range correspondingly. This happens only if you have not established your own values. When you want the console device to manage the size of the window or the text range by itself, you must call the commands with statements of values, for example "<CSI>t".

SET LINE WIDTH ("<CSI><width>u")

Specifies the number of characters per line. You can use the remaining space for graphics.

SET LEFT OFFSET ("<CSI><offset>x")

Specifies the starting vertical raster line (not column) at which the text range should start. For example,"<CSI>8x" uses eight lines of space in the left window margin for small graphics, scroll bars, etc.

SET TOP OFFSET ("<CSI><offset>y")

Specifies the starting horizontal raster line (not text line) at which the text range should start. The remaining space can be used for displaying graphics, tab positions, etc.

CURSOR ON ("<CSI>0 p")

Enables cursor.

CURSOR OFF ("<CSI> p")

Disables cursor.

WINDOW STATUS ("<CSI>o q")

Sends window status request. The system returns a status report in the following form, readable using CMD_READ:

```
"<CSI>1;1;,<bottom border>;right border> r"
```

The window status report returns the positions of the upper left corner (1,1) and the lower right corner.

RAW EVENTS

Requests additional information about the system from the console device. For this you must inform the console device which RAW EVENTS you want to see:

0	no operation
1	keypress and the release of the key
2	mouse button pressed
3	window was activated
4	mouse pointer position
5	unused
6	timer events

7	gadget clicked
8	gadget released
9	requester displayed
10	menu selected
11	close gadget clicked
12	window resized
13	window redrawn
14	Preferences changed
15	disk removed
16	disk inserted

The RAW EVENTS read through the console device are identical to those events accessible from the input device. Here too the corresponding flags for checking the event must be set. The following syntax sends the required events:

```
<CSI>Event number{
```

The events return in the following format:

```
"<CSI><Class>;<SubClass>;<KeyCode>;<Qualifiers>;<X>;<Y>;<Seconds
>;<MicroSeconds>|"
```

When you look at the input event structure, you determine large coincidences between this structure and the control string, which are sanded from the console device. Each decimal string means the following:

<Class> Returns the number of results received from the console device. You can check multiple results of all of the results from the console device. For example, if you want keypresses and mouse button clicks to be returned, you can access them through "<CSI>1{" followed by "<CSI>2{", or simply through "<CSI>1;2{".

<SubClass> Most often contains the value 0. SubClass contains the value 1 only when the mouse is plugged into gameport 2.

<KeyCode> Contains the decimal string that indicates the keyboard code of the pressed or released key (see Section 4.8).

<Qualifiers>

Indicates which of the <Ctrl>, <Alt> and <Shift> keys was pressed.

1	left <Shift>
2	right <Shift>
4	<Caps lock>
8	<Ctrl>
16	left <Alt>
32	right <Alt>

64	left <Amiga>
128	right <Amiga>
256	numeric keypad key
512	key repeated (repeat function)
1024	interrupt (unused)
2048	multi broadcast (result for current window)
4096	left mouse button
8192	middle mouse button (unused)
16384	right mouse button
32768	relative mouse movement

When multiple `Qualifiers` are pressed, the corresponding values are added and given in `<Qualifier>`.

`<X>` `<Y>` Returns the relative mouse movement or the address of the chosen gadget (X<<16+Y).

`<Seconds>` Returns the system time in seconds.

`<Microseconds>`
Returns the system time in microseconds.

The user must decode the string supplied by the console device and interpret the values found there. It takes little time to calculate how much more he gets directly though the IDCMP flags of the events and the keyboard code.

Now that we've described all of the control strings that can be sent and received, we have here a short application that you can enter, compile, link and run. This console device editor allows you to enter control strings from the keyboard and watch the result firsthand. Because the control sequence can be chosen by sources other than through the keyboard, we check the escape key and interpret Escape as <CSI>. You can get out of the editor by entering <q><Return>.

Combine all of the Con_Support routines presented in this chapter to form the Con_Support.c program. Compile the Con_Support.c program and link it to the following program. Don't forget the proper include files in the Con_Support.c file (exec/types.h, exec/memory.h, exec/devices.h, devices/console.h, devices/keymap.h)

```
/*****************************************************************
*                            Conni.c                              *
*                      (c) Bruno Jennrich                         *
*                          August 1988                            *
*****************************************************************/
/*****************************************************************
* Compile-Info:                                                   *
* cc Conni.c                                                      *
* ln Conni.o Con_Support.o Devs_Support.o -lc                     *
*****************************************************************/
```

```
#include "exec/types.h"
#include "exec/memory.h"
#include "exec/devices.h"
#include "devices/console.h"
#include "devices/keymap.h"
#include "intuition/intuitionbase.h"
#include "intuition/intuition.h"

#define MEMTYPE   (MEMF_CHIP | MEMF_CLEAR)
#define CON_LEN (ULONG) (sizeof (struct IOStdReq))

VOID *Open();
VOID *AllocMem();
VOID *GetDeviceBlock();
VOID *OpenLibrary();
VOID *OpenScreen();
VOID *OpenWindow();

struct Screen        *Screen        = 0l;
struct Window        *Window        = 0l;
struct IntuitionBase *IntuitionBase = 0l;

struct NewScreen      NewScreen = {
                                   0,0,640,200,4,
                                   0,1,
                                   HIRES,
                                   CUSTOMSCREEN,
                                   0l,
                                   (UBYTE*) "No Name",
                                   0l,
                                   0l
                                };

struct NewWindow      NewWindow = {
                                   0,0,
                                   640,200,
                                   0,1,
                                   0l,
                                   (ULONG) ACTIVATE,
                                   0l,
                                   0l,
                                   (UBYTE*) "Console-Device-
Editor (c) Bruno Jennrich",
                                   0l,
                                   0l,
                                   0,0,
                                   0,0,
                                   CUSTOMSCREEN
                                };

struct IOStdReq  *ConsoleRead  = 0l,
                 *ConsoleWrite = 0l;

/***************************************************************
*                     CloseIt()                         (User)*
*                                                             *
* Function: In case of erro rclose everything                 *
*-------------------------------------------------------------*
* Input - Parameter:                                          *
*                                                             *
* String: Error-Message                                       *
***************************************************************/
```

```
VOID CloseIt (String)
char         *String;
{
   UWORD i;
   UWORD *dff180 = (UWORD *)0xdff180;
   UWORD Error = 0;

   if (strlen (String) > 0l)
   {
      for (i=0;i<0xffff;i++)  *dff180 = i;
      puts (String);
      Error = 10;
   }

   if (Window != 0l)         CloseWindow (Window);
   if (Screen != 0l)         CloseScreen (Screen);
   if (IntuitionBase != 0l) CloseLibrary (IntuitionBase);

   if (ConsoleRead != 0)    Close_A_Device (ConsoleRead);
   if (ConsoleWrite != 0)   FreeDeviceBlock (ConsoleWrite);
   exit (Error);
}


/****************************************************************
*                    Open_Screen_and_Window()            (User)*
*                                                               *
* Function: Open editor screen and window                       *
****************************************************************/

VOID Open_Screen_and_Window()
{
   IntuitionBase = (struct IntuitionBase*)
                   OpenLibrary ("intuition.library",0l);
   if (IntuitionBase == 0l) CloseIt ("No IntuitionBase !");

   Screen = (struct Screen *) OpenScreen (&NewScreen);
   if (Screen == 0l) CloseIt ("No Screen !");

   NewWindow.Screen = Screen;

   Window = (struct Window *) OpenWindow (&NewWindow);
   if (Window == 0l) CloseIt ("No Window !");
}


/****************************************************************
*                    Close_Screen_and_Window()           (User)*
*                                                               *
* Function: Close editor screen and window                      *
****************************************************************/

VOID Close_Screen_and_Window()
{
   CloseWindow  (Window);
   CloseScreen  (Screen);
   CloseLibrary (IntuitionBase);
}
```

```
/***************************************************************
*                        main()                          (User)*
*                                                              *
***************************************************************/
main ()
{
   UWORD  i;
   BYTE   InputString[256];
   BYTE  *BufPointer;       /* Actual input position inside */
                            /* of InputString               */
   UWORD  Pos;              /* Number chars in InputString  */
   BOOL   Quit   = FALSE;   /* Programm ended ? */
   BOOL   Return = FALSE;   /* Return pressed ? */

   Open_Screen_and_Window();

   ConsoleRead  = (struct IOStdReq *)GetDeviceBlock (CON_LEN);
   ConsoleWrite = (struct IOStdReq *)GetDeviceBlock (CON_LEN);

   ConsoleRead->io_Data    = (APTR) Window;
   ConsoleRead->io_Length  = (ULONG) (sizeof (struct Window));

   Open_A_Device ("console.device",0l,&ConsoleRead,0l,0l);

   Console_Copy (ConsoleRead,ConsoleWrite);

   BufPointer = InputString;
   Pos = 0;
   while (!Quit)
   {
   while (!Return)
   {
         *BufPointer = (BYTE)0;
         Console_Read (ConsoleRead,BufPointer,1l);
         if (*BufPointer == (BYTE)0x08)  /* Backspace */
         {
            if (Pos>0)
            {
               *BufPointer = (BYTE)0;
               BufPointer--;
               if (*BufPointer == (BYTE) 0x1b)
               {                            /* <   C   S   I  > */
                  Console_Write
(ConsoleWrite,"\010\010\010\010\010",-1l);
                  Console_Write (ConsoleWrite,"     ",-1l);
                  Console_Write
(ConsoleWrite,"\010\010\010\010\010",-1l);
               }
               else
               {
                  Console_Write (ConsoleWrite,"\010",1l);
                  Console_Write (ConsoleWrite," ",1l);
                  Console_Write (ConsoleWrite,"\010",1l);
               }
               *BufPointer = (BYTE)0;
               Pos--;
            }
         }
         else
         {
            if (Pos <256)
            {
```

```
                if (*BufPointer == (BYTE) 0x9b)
                {           /* Replace revieved CSI with 0x07 */
                   *BufPointer = 0x7;
                }
                if (*BufPointer == 0x0d) /* Return */
                {
                   Return = TRUE;
                   Console_Write (ConsoleWrite,"\012",1l);
                   if (*InputString == 'q') Quit = TRUE;
                                  /* 'q' pressed ? */
                }
                else
                if (*BufPointer == 0x1b) /* Escape */
                {
                   Console_Write (ConsoleWrite,"<CSI>",5l);
                }
                else Console_Write (ConsoleWrite,BufPointer,1l);

                BufPointer++;
                Pos++;
             }
          }
       }
       Return = FALSE;
       *BufPointer = (BYTE)0;

       if (*InputString == (BYTE)0x1b)
       {   /* Display control string after Return */
          *InputString = (BYTE) 0x9b;
          Console_Write (ConsoleWrite,InputString,-1l);
       }
       BufPointer = InputString;
       Pos = 0;
    }

    Close_A_Device (ConsoleRead);
    FreeDeviceBlock (ConsoleWrite);

    Close_Screen_and_Window();
}
```

Note: Once you enter "<CSI>1{" there is no going back.

The <CSI> codes are replaced with BELLs (0x07) when the control strings are received. Many of the keypresses (e.g., cursor keys) also send a <CSI>. Therefore, cursor keys and function keys may not work. One last item of interest: The control codes listed above also function in a CON: window. To see if the ConIn program works press the <Esc> <0> <p> to turn off the cursor, then <Esc> <p> to turn is back on.

4.9.1 Key mapping

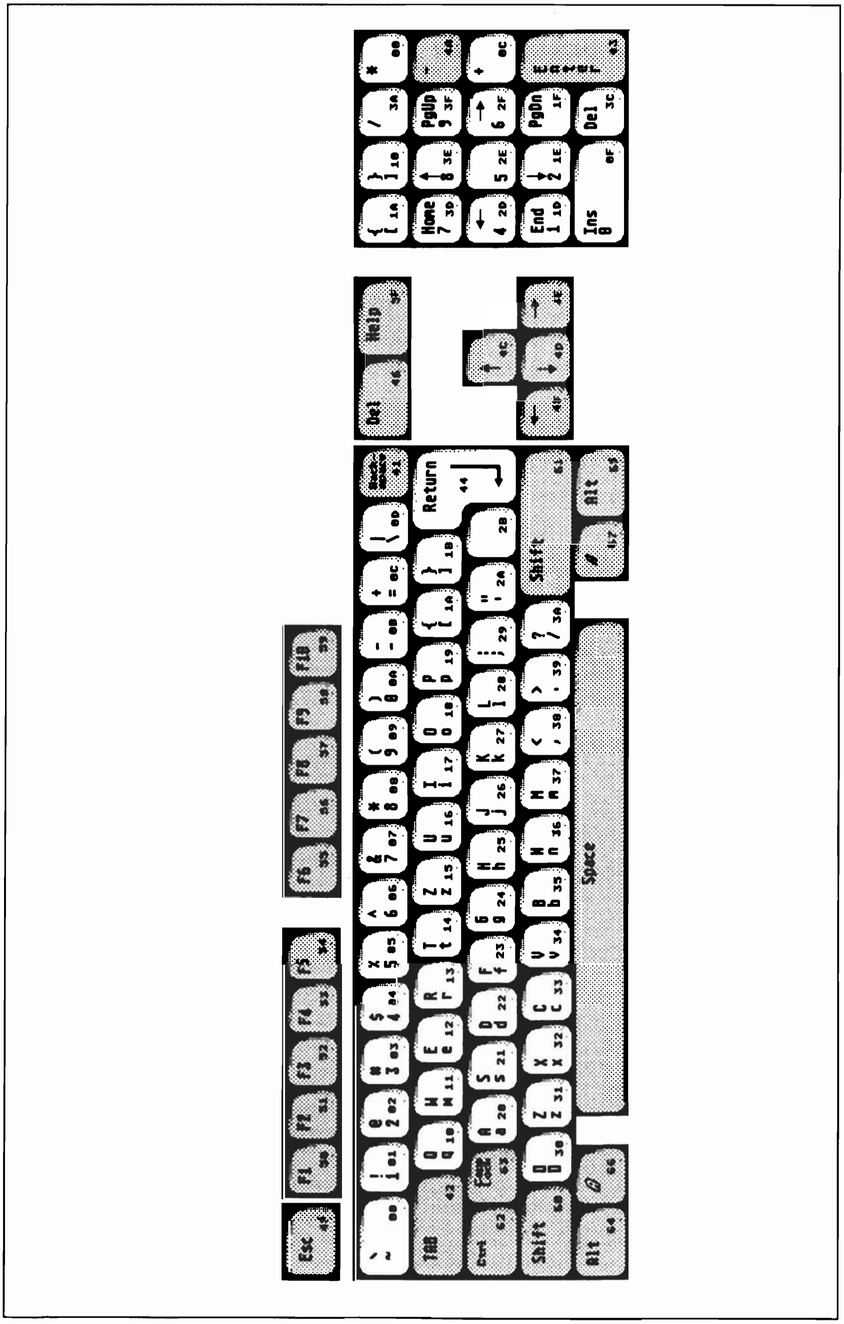
The console device allows the option of overlaying new keys. This makes it possible to print a string like "Hello people, how are you?" in response to a single keypress. The console device gets the key code of the character that was pressed from the input device by way of the keyboard device. The console then extracts the ASCII code from the tables that correspond to the key that was pressed and gives this to the user. You can change this table, provided you know the keymap structure:

```
Offset    Structure
------    --------
          struct KeyMap
          {
 0  0x00     UBYTE *km_LoKeyMapTypes;
 4  0x04     ULONG *km_LoKeyMap;
 8  0x08     UBYTE *km_LoCapsable;
12  0x0c     UBYTE *km_LoRepeatable;
16  0x10     UBYTE *km_HiKeyMapTypes;
20  0x14     ULONG *km_HiKeyMap;
24  0x18     UBYTE *km_HiCapsable;
28  0x1c     UBYTE *km_HiRepeatable;
32  0x20  } /* defined in "devices/keymaps.h" */
```

This structure contains the pointer to memory ranges that compare what was sent with a certain keypress in the console device. The difference between Hi and Lo map is important. The Lo keymap contains the data for key codes 0x0 through 0x3f. The Hi keymap contains the data for key codes 0x40-0x67.

Next we need the `KeyMapTypes` and the `KeyMap`. The pointers `km_LoKeyMapTypes` and `km_HiKeyMapTypes` point to a byte array which determines which qualifier (<Shift>, <Alt>, <Ctrl>) is supported by the key, or if a string (`KCF_STRING`) should be sent instead of a simple ASCII code. The following values are allowed:

```
#define KC_NOQUAL   0x00
#define KCF_SHIFT   0x01
#define KCF_ALT     0x02
#define KCF_CONTROL 0x04
#define KC_VANILLA  0x07 /* Shift+Alt+Ctrl */
#define KCF_STRING  0x40 /* String */
/* defined in "devices/keymaps.h" */
```



The ASCII codes that should be sent if a single key or a key with a qualifier is pressed are saved in `km_LoKeyMap` and `km_HiKeyMap`. This memory area is a long word array (4 bytes = 4 ASCII codes).

The result is that a key can support only two qualifiers (e.g., <Shift><Alt> or <Ctrl><Alt>). ASCII codes are sent when a single key is pressed, when a key is pressed in conjunction with one of the qualifiers (e.g., <Shift> or <Alt>), and when a key is pressed in conjunction with two qualifiers (e.g., <Shift><Alt>).

By using strings instead of simpler ASCII codes, up to eight different strings can be sent out based on a single key and the corresponding qualifier. Let's look at a German keymap that makes simple ASCII codes available as strings:

```
#asm
;ownkeymap.asm (c) Bruno Jennrich

KC_NOQUAL   equ 0
KC_VANILLA  equ 7
KCF_SHIFT   equ 1
KCF_ALT     equ 2
KCF_CONTROL equ 4
KCF_STRING  equ 64
CSI         equ $9b

   public _LoKeyMapTypes
   even
_LoKeyMapTypes:
   dc.b KC_VANILLA          ;$00    Tilde
   dc.b KC_VANILLA          ;$01    1
   dc.b KC_VANILLA          ;$02    2
   dc.b KC_VANILLA          ;$03    3
   dc.b KC_VANILLA          ;$04    4
   dc.b KC_VANILLA          ;$05    5
   dc.b KC_VANILLA          ;$06    6
   dc.b KC_VANILLA          ;$07    7
   dc.b KC_VANILLA          ;$08    8
   dc.b KC_VANILLA          ;$09    9
   dc.b KC_VANILLA          ;$0a    0
   dc.b KC_VANILLA          ;$0b    _
   dc.b KC_VANILLA          ;$0c    '
   dc.b KC_VANILLA          ;$0d    \
   dc.b 0                   ;$0e    undefinied   !!!!!!!
   dc.b 0                   ;$0e    undefinied
   dc.b KC_NOQUAL           ;$0f    0 (Number field)
   dc.b KC_VANILLA          ;$10    q
   dc.b KC_VANILLA          ;$11    w
   dc.b KC_VANILLA          ;$12    e
   dc.b KC_VANILLA          ;$13    r
   dc.b KC_VANILLA          ;$14    t
   dc.b KC_VANILLA          ;$15    z , y on US keyboard
   dc.b KC_VANILLA          ;$16    u
   dc.b KC_VANILLA          ;$17    i
   dc.b KC_VANILLA          ;$18    o
   dc.b KC_VANILLA          ;$19    p
   dc.b KC_VANILLA          ;$1a    |
   dc.b KC_VANILLA          ;$1b    +
```

```
        dc.b 0                      ;$1c     undefinied
        dc.b KC_NOQUAL              ;$1d     1 (Number field)
        dc.b KC_NOQUAL              ;$1e     2 (Number field)
        dc.b KC_NOQUAL              ;$1f     3 (Number field)
        dc.b KC_VANILLA             ;$20     a
        dc.b KC_VANILLA             ;$21     s
        dc.b KC_VANILLA             ;$22     d
        dc.b KC_VANILLA             ;$23     f
        dc.b KC_VANILLA             ;$24     g
        dc.b KC_VANILLA             ;$25     h
        dc.b KC_VANILLA             ;$26     j
        dc.b KC_VANILLA             ;$27     k
        dc.b KC_VANILLA             ;$28     l
        dc.b KC_VANILLA             ;$29     v
        dc.b KC_VANILLA             ;$2a     d
        dc.b 0                      ;$2b     reserved
        dc.b 0                      ;$2c     undefinied
        dc.b KC_NOQUAL              ;$2d     4 (Number field)
        dc.b KC_NOQUAL              ;$2e     5 (Number field)
        dc.b KC_NOQUAL              ;$2f     6 (Number field)
        dc.b 0                      ;$30     reserved
        dc.b KC_VANILLA             ;$31     y, z on US keyboard
        dc.b KC_VANILLA             ;$32     x
        dc.b KC_VANILLA             ;$33     c
        dc.b KC_VANILLA             ;$34     v
        dc.b KC_VANILLA             ;$35     b
        dc.b KC_VANILLA             ;$36     n
        dc.b KC_VANILLA             ;$37     m
        dc.b KC_VANILLA             ;$38     ,
        dc.b KC_VANILLA             ;$39     .
        dc.b KC_VANILLA             ;$3a     -
        dc.b 0                      ;$3b     undefinied
        dc.b KC_NOQUAL              ;$3c     , (Number field)
        dc.b KC_NOQUAL              ;$3d     7 (Number field)
        dc.b KC_NOQUAL              ;$3e     8 (Number field)
        dc.b KC_NOQUAL              ;$3f     9 (Number field)

        public _LoKeyMap
        even
_LoKeyMap:
        dc.b "~","'","]","["                 ;$00     [
        dc.b $21+$80,$31+$80,"!","1"         ;$01     1
        dc.b $22+$80,$32+$80,$22,"2"         ;$02     2
        dc.b $a7+$80,$33+$80,"'","3"         ;$03     3
        dc.b $24+$80,$34+$80,"$","4"         ;$04     4
        dc.b $25+$80,$35+$80,"%","5"         ;$05     5
        dc.b $26+$80,$36+$80,"&","6"         ;$06     6
        dc.b $2f+$80,$37+$80,"/","7"         ;$07     7
        dc.b $28+$80,$38+$80,"(","8"         ;$08     8
        dc.b $29+$80,$39+$80,")","9"         ;$09     9
        dc.b $3d+$80,$30+$80,"=","0"         ;$0a     0
        dc.b $3f+$80,$df+$80,"?","_"         ;$0b     _
        dc.b $27+$80,$60+$80,"'","`"         ;$0c     '
        dc.b $7c+$80,$5c+$80,"|","\"         ;$0d     \
        dc.l $0                              ;$0e     undefinied
        dc.l $0                              ;$0e     undefinied
        dc.b $00,$00,$00,"0"                 ;$0f     0 (Number field)
        dc.b $51+$80,$71+$80,"Q","q"         ;$10     q
        dc.b $57+$80,$77+$80,"W","w"         ;$11     w
        dc.b $45+$80,$65+$80,"E","e"         ;$12     e
        dc.b $52+$80,$72+$80,"R","r"         ;$13     r
        dc.b $54+$80,$74+$80,"T","t"         ;$14     t
```

```
    dc.b $5a+$80,$7a+$80,"Z","z"        ;$15     z, y on US
    dc.b $55+$80,$75+$80,"U","u"        ;$16     u
    dc.b $49+$80,$69+$80,"I","i"        ;$17     i
    dc.b $4f+$80,$6f+$80,"O","o"        ;$18     o
    dc.b $50+$80,$70+$80,"P","p"        ;$19     p
    dc.b $dc+$80,$fc+$80,"\","|"        ;$1a     |
    dc.b $2a+$80,$2b+$80,"*","+"        ;$1b     +
    dc.b $00,$00,$00,$00                ;$1c     undefinied
    dc.b $00,$00,$00,"1"                ;$1d     1 (Number field)
    dc.b $00,$00,$00,"2"                ;$1e     2 (Number field)
    dc.b $00,$00,$00,"3"                ;$1f     3 (Number field)
    dc.b $41+$80,$61+$80,"A","a"        ;$20     a
    dc.b $53+$80,$73+$80,"S","s"        ;$21     s
    dc.b $44+$80,$64+$80,"D","d"        ;$22     d
    dc.b $46+$80,$66+$80,"F","f"        ;$23     f
    dc.b $47+$80,$67+$80,"G","g"        ;$24     g
    dc.b $48+$80,$68+$80,"H","h"        ;$25     h
    dc.b $4a+$80,$6a+$80,"J","j"        ;$26     j
    dc.b $4b+$80,$6b+$80,"K","k"        ;$27     k
    dc.b $4c+$80,$6c+$80,"L","l"        ;$28     l
    dc.b $d6+$80,$f6+$80,"V","v"        ;$29     v
    dc.b $c4+$80,$e4+$80,"D","d"        ;$2a     d
    dc.b $00,$00,$00,$00                ;$2b     reservied
    dc.b $00,$00,$00,$00                ;$2c     undefined
    dc.b $00,$00,$00,"4"                ;$2d     4 (Number field)
    dc.b $00,$00,$00,"5"                ;$2e     5 (Number field)
    dc.b $00,$00,$00,"6"                ;$2f     6 (Number field)
    dc.b $00,$00,$00,$00                ;$30     reserved
    dc.b $59+$80,$79+$80,"Y","y"        ;$31     y, z on US
    dc.b $58+$80,$78+$80,"X","x"        ;$32     x
    dc.b $43+$80,$63+$80,"C","c"        ;$33     c
    dc.b $56+$80,$76+$80,"V","v"        ;$34     v
    dc.b $42+$80,$62+$80,"B","b"        ;$35     b
    dc.b $4e+$80,$6e+$80,"N","n"        ;$36     n
    dc.b $4f+$80,$6f+$80,"M","m"        ;$37     m
    dc.b $3b+$80,$2c+$80,";",","        ;$38     ,
    dc.b $3a+$80,$2e+$80,":","."        ;$39     .
    dc.b $5f+$80,$2d+$80,"_","-"        ;$3a     -
    dc.b $00,$00,$00,$00                ;$3b     undefined
    dc.b $00,$00,$00,","                ;$3c     , (Number field)
    dc.b $00,$00,$00,"1"                ;$3d     7 (Number field)
    dc.b $00,$00,$00,"2"                ;$3e     8 (Number field)
    dc.b $00,$00,$00,"3"                ;$3f     9 (Number field)

    public _HiKeyMapTypes
    even
_HiKeyMapTypes:
    dc.b KC_NOQUAL              ;$40     Space
    dc.b KC_NOQUAL              ;$41     BackSpace
    dc.b KC_NOQUAL              ;$42     Tab
    dc.b KC_NOQUAL              ;$43     Enter
    dc.b KC_NOQUAL              ;$44     Return
    dc.b KC_NOQUAL              ;$45     Escape
    dc.b KC_NOQUAL              ;$46     Delete
    dc.b 0                      ;$47     undefined
    dc.b 0                      ;$48     undefined
    dc.b 0                      ;$49     undefined
    dc.b KC_NOQUAL              ;$4a     Number feild
    dc.b 0                      ;$4b     undefined
    dc.b KCF_STRING+KCF_SHIFT   ;$4c     Up Arrow
    dc.b KCF_STRING+KCF_SHIFT   ;$4d     Down Arrow
    dc.b KCF_STRING+KCF_SHIFT   ;$4e     Forward Arrow
```

```
    dc.b KCF_STRING+KCF_SHIFT   ;$4f     Backward Arrow
    dc.b KCF_STRING+KCF_SHIFT   ;$50     F1
    dc.b KCF_STRING+KCF_SHIFT   ;$51     F2
    dc.b KCF_STRING+KCF_SHIFT   ;$52     F3
    dc.b KCF_STRING+KCF_SHIFT   ;$53     F4
    dc.b KCF_STRING+KCF_SHIFT   ;$54     F5
    dc.b KCF_STRING+KCF_SHIFT   ;$55     F6
    dc.b KCF_STRING+KCF_SHIFT   ;$56     F7
    dc.b KCF_STRING+KCF_SHIFT   ;$57     F8
    dc.b KCF_STRING+KCF_SHIFT   ;$58     F9
    dc.b KCF_STRING+KCF_SHIFT   ;$59     F10
    dc.b 0                      ;$5a     undefined
    dc.b 0                      ;$5b     undefined
    dc.b 0                      ;$5c     undefined
    dc.b 0                      ;$5d     undefined
    dc.b 0                      ;$5e     undefined
    dc.b KCF_STRING             ;$5f     Help
    dc.b 0                      ;$60     SHIFT left
    dc.b 0                      ;$61     SHIFT right
    dc.b 0                      ;$62     CAPS LOCK
    dc.b 0                      ;$63     CTRL
    dc.b 0                      ;$64     ALT left
    dc.b 0                      ;$65     ALT right
    dc.b 0                      ;$66     AMIGA left
    dc.b 0                      ;$67     AMIGA right

    public _HiKeyMap
    even
_HiKeyMap:

    dc.b $00,$00,$00,$20      ;$40  Space
    dc.b $00,$00,$00,$08      ;$41  BackSpace
    dc.b $00,$00,$00,$09      ;$42  Tab
    dc.b $00,$00,$00,$0d      ;$43  Enter
    dc.b $00,$00,$00,$0d      ;$44  Return
    dc.b $00,$00,$9b,$1b      ;$45  Escape
    dc.b $00,$00,$00,$7f      ;$46  Delete
    dc.b $00,$00,$00,$00      ;$47  undefined
    dc.b $00,$00,$00,$00      ;$48  undefined
    dc.b $00,$00,$00,$00      ;$49  undefined
    dc.b $00,$00,$00,"-"      ;$4a  Numeric Pad
    dc.b $00,$00,$00,$00      ;$4b  undefined
    dc.l Up_Arrow             ;$4c  Up Arrow
    dc.l Down_Arrow           ;$4d  Down Arrow
    dc.l Forward_Arrow        ;$4e  Forward Arrow
    dc.l Backward_Arrow       ;$4f  Backward Arrow
    dc.l F1                   ;$50  F1
    dc.l F2                   ;$51  F2
    dc.l F3                   ;$52  F3
    dc.l F4                   ;$53  F4
    dc.l F5                   ;$54  F5
    dc.l F6                   ;$55  F6
    dc.l F7                   ;$56  F7
    dc.l F8                   ;$57  F8
    dc.l F9                   ;$58  F9
    dc.l F10                  ;$59  F10
    dc.b $00,$00,$00,$00      ;$5a  undefined
    dc.b $00,$00,$00,$00      ;$5b  undefined
    dc.b $00,$00,$00,$00      ;$5c  undefined
    dc.b $00,$00,$00,$00      ;$5d  undefined
    dc.b $00,$00,$00,$00      ;$5e  undefined
    dc.l Help                 ;$5f  Help
```

```
   dc.b $00,$00,$00,$00       ;$60  SHIFT left
   dc.b $00,$00,$00,$00       ;$61  SHIFT right
   dc.b $00,$00,$00,$00       ;$62  CAPS LOCK
   dc.b $00,$00,$00,$00       ;$63  CTRL
   dc.b $00,$00,$00,$00       ;$64  ALT left
   dc.b $00,$00,$00,$00       ;$65  ALT right
   dc.b $00,$00,$00,$00       ;$66  AMIGA left
   dc.b $00,$00,$00,$00       ;$67  AMIGA right


Up_Arrow:
   dc.b  2                    ; Length of the strings (unshifted)
   dc.b  Up_Arrow_UnShift-Up_Arrow   ; Offset
   dc.b  2                    ; Length of the strings (shifted)
   dc.b  Up_Arrow_Shift-Up_Arrow     ; Offset
Up_Arrow_UnShift:
   dc.b CSI,"A"
Up_Arrow_Shift:
   dc.b CSI,"T"

Down_Arrow:
   dc.b  2                    ; Length of the strings (unshifted)
   dc.b  Down_Arrow_UnShift-Down_Arrow    ; Offset
   dc.b  2                    ; Length of the strings (shifted)
   dc.b  Down_Arrow_Shift-Down_Arrow      ; Offset
Down_Arrow_UnShift:
   dc.b CSI,"B"
Down_Arrow_Shift:
   dc.b CSI,"S"

Forward_Arrow:
   dc.b  2                    ; Length of the strings (unshifted)
   dc.b  Forward_Arrow_UnShift-Forward_Arrow ; Offset
   dc.b  3                    ; Length of the strings (shifted)
   dc.b  Forward_Arrow_Shift-Forward_Arrow   ; Offset
Forward_Arrow_UnShift:
   dc.b CSI,"C"
Forward_Arrow_Shift:
   dc.b CSI," A"

Backward_Arrow:
   dc.b  2                    ; Length of the strings (unshifted)
   dc.b  Backward_Arrow_UnShift-Backward_Arrow    ; Offset
   dc.b  3                     ; Length of the strings (shifted)
   dc.b  Backward_Arrow_Shift-Backward_Arrow      ; Offset
Backward_Arrow_UnShift:
   dc.b CSI,"D"
Backward_Arrow_Shift:
   dc.b CSI," @"

F1:
   dc.b  3                ; Length of the strings (unshifted)
   dc.b  F1_UnShift-F1    ; Offset
   dc.b  4                ; Length of the strings (shifted)
   dc.b  F1_Shift-F1      ; Offset
F1_UnShift:
   dc.b CSI,"0~"
F1_Shift:
   dc.b CSI,"10~"

F2:
   dc.b  3                ; Length of the strings (unshifted)
```

```
    dc.b  F2_UnShift-F2     ; Offset
    dc.b  4                 ; Lenght of the strings (shifted)
    dc.b  F2_Shift-F2       ; Offset
F2_UnShift:
    dc.b CSI,"1~"
F2_Shift:
    dc.b CSI,"11~"

F3:
    dc.b  3                 ; Length of the strings (unshifted)
    dc.b  F3_UnShift-F3     ; Offset
    dc.b  4                 ; Length of the strings (shifted)
    dc.b  F3_Shift-F3       ; Offset
F3_UnShift:
    dc.b CSI,"2~"
F3_Shift:
    dc.b CSI,"12~"

F4:
    dc.b  3                 ; Length of the strings (unshifted)
    dc.b  F4_UnShift-F4     ; Offset
    dc.b  4                 ; Length of the strings (shifted)
    dc.b  F4_Shift-F4       ; Offset
F4_UnShift:
    dc.b CSI,"3~"
F4_Shift:
    dc.b CSI,"13~"

F5:
    dc.b  3                 ; Length of the strings (unshifted)
    dc.b  F5_UnShift-F5     ; Offset
    dc.b  4                 ; Length of the strings (shifted)
    dc.b  F5_Shift-F5       ; Offset
F5_UnShift:
    dc.b CSI,"4~"
F5_Shift:
    dc.b CSI,"14~"

F6:
    dc.b  3                 ; Length of the strings (unshifted)
    dc.b  F6_UnShift-F6     ; Offset
    dc.b  4                 ; Length of the strings (shifted)
    dc.b  F6_Shift-F6       ; Offset
F6_UnShift:
    dc.b CSI,"5~"
F6_Shift:
    dc.b CSI,"15~"

F7:
    dc.b  3                 ; Length of the strings (unshifted)
    dc.b  F7_UnShift-F7     ; Offset
    dc.b  4                 ; Length of the strings (shifted)
    dc.b  F7_Shift-F7       ; Offset
F7_UnShift:
    dc.b CSI,"6~"
F7_Shift:
    dc.b CSI,"16~"

F8:
    dc.b  3                 ; Length of the strings (unshifted)
    dc.b  F8_UnShift-F8     ; Offset
    dc.b  4                 ; Length of the strings (shifted)
```

```
   dc.b  F8_Shift-F8       ; Offset
F8_UnShift:
   dc.b CSI,"7~"
F8_Shift:
   dc.b CSI,"17~"

F9:
   dc.b  3                 ; Length of the strings (unshifted)
   dc.b  F9_UnShift-F9     ; Offset
   dc.b  4                 ; Length of the strings (shifted)
   dc.b  F9_Shift-F9       ; Offset
F9_UnShift:
   dc.b CSI,"8~"
F9_Shift:
   dc.b CSI,"18~"

F10:
   dc.b  3                 ; Length of the strings (unshifted)
   dc.b  F10_UnShift-F10   ; Offset
   dc.b  4                 ; Length of the strings (shifted)
   dc.b  F10_Shift-F10     ; Offset
F10_UnShift:
   dc.b CSI,"9~"
F10_Shift:
   dc.b CSI,"19~"

Help:
   dc.b  3                    ; Length of the strings (unshifted)
   dc.b  Help_UnShift-Help    ; Offset
Help_UnShift:
   dc.b CSI,"?~"
   even
#endasm
```

Note: For unknown reasons we must use \$0e twice for the keyboard code instead of one byte in the `KeyMapTypes` table and eight bytes instead of four in the `Keymap` table. If you don't do this, all of the keypresses end with incorrect codes. For example, "v" instead of C or "m" instead of M.

Remember that the `KeyMap` and `KeyMapTypes` arrays begin in word addresses (even). After determination of the arrays you should also make sure that the rest of the program continues with an even address otherwise a Guru Meditation occurs. This Guru informs you of an address error (Guru number \$00000003).

Now let's examine the entries in `LoKeyMap` more closely: you have determined that the first byte contains the value `"A"+$80` for the keyboard code \$20. The second byte contains `"a"+$80` and the last two bytes contain the value "A" and "a". Unfortunately the Aztec C assembler does not understand expressions like `"A"+$80`. That is why we have translated the character "A" into its ASCII code \$41. Let's see which of the four ASCII codes are sent when a key is pressed in conjunction with a qualifier:

allowable qualifiers (keyMapTypes)

printable qualifiers		S	A	C	S+A	C+A	S+C	S+A+C
	a	a	a	a	a	a	a	a
S	a	A	a	a	A	a	A	a
A	a	a	A	a	a+ $80	A	a	a+ $80
C	a	a	a	A	a	a+ $80	a+ $80	a~ $30
S+A	a	A	A	a	A+ $80	A	A	A+ $80
C+A	a	a	A	A	a+ $80	A+ $80	a+ $80	a+ $80 ~$30
S+C	a	A	a	A	A	a+ $80	A+ $80	A~ §30
S+A+C	a	A	A	A	A+ $80	A+ $80	A+ $80	A+ $80 ~$30

S=Shift
A=Alt
C=Control

Example: allowable qualifier: S+A
: printable qualifier: C+A
Result: "a"+$80

KeyMapEntry:
dc.b "A"+$80, "a"+$80, "A", "a"

You see that one of the ASCII codes "a", "A", "A"+$80, or "a"+$80 is sent only when the qualifiers <Alt> and <Shift> are pressed. When all three qualifiers are allowed, bits 5 and 6 ($30) of the sent code are cleared when the <Ctrl> key is pressed. You also have no option of testing for the ASCII code sent in conjunction with <Ctrl>, independently from the codes established in the keymap.

This changes somewhat when we use strings instead of simple ASCII codes. Here you must bear in mind that the four bytes in the keymap act as an entry point to one or more string descriptors. Such a string descriptor has the following format:

1.) byte length of the string to be displayed
2.) byte offset of the string at the beginning of the descriptor

Here is an example:

```
StringDescriptor:
   dc.b 8                                     ;length
   dc.b Stringtobedisplayed1-StringDescriptor ;Offset
   dc.b 14                                    ;length
   dc.b Stringtobedisplayed2-StringDescriptor ;Offset
Stringtobedisplayed1: dc.b "String 1"
Stringtobedisplayed2: dc.b "second String"
```

Because the strings to be displayed are addressed over offsets, the strings must be placed in the range from +127 to -128 bytes from the beginning of the string descriptor. Now you can represent all three qualifiers and their combinations through other strings. The illustration on the opposite page tells which combination of allowed and pressed qualifiers display which string.

Remember that with one allowable qualifier, two strings must be available; for two allowable qualifiers, four strings must be available; and with three qualifiers, eight strings must be available. A string descriptor must also be specified for each of the strings to be displayed. Now the pointers `Lo/HiCapsable` and `Lo/HiRepeatable` must be explained.

Certain keys like the <Caps lock> key are not represented by their shifted values. So <Caps lock> displays a "1" instead of a "!". The `CapsAble` pointer points to a 8 byte array from which a bit is responsible for seeing if a key should execute Caps lock (Bit == 1) or ignore it (Bit == 0). The `CapsAble` bit 0 of the first byte is set to zero for the key number zero (`LoCapsable`). The first bit of the second byte pertains to key 8, and so on. The first bit from `HiCapsable` pertains to key 0x40. Two pointers point to a 64 bit = byte size array.

allowable qualifiers (keyMapTypes)

printable qualifiers		S	A	C	S+A	C+A	S+C	S+A+C
	A	A	A	A	A	A	A	A
S	A	B	A	A	B	A	B	B
A	A	A	B	A	C	B	A	C
C	A	A	A	B	A	C	C	E
S+A	A	B	B	A	D	D	B	D
C+A	A	A	B	B	C	B	C	G
S+C	A	B	A	B	A	C	D	F
S+A+C	A	B	B	B	D	D	D	H

S=Shift
A=Alt
C=Control

KeyMapEntry

```
NewA:
    dc . b 1
    dc . b A-NewA
    dc . b 1
    dc . b B-NewA
    dc . b 1
    dc . b C-NewA
    dc . b 1
    dc . b D-NewA
    dc . b 1
    dc . b E-NewA
    dc . b 1
    dc . b F-NewA
    dc . b 1
    dc . b G-NewA
    dc . b 1
    dc . b H-NewA
```

```
A: dc . b "A"
B: dc . b "B"
C: dc . b "C"
D: dc . b "D"
E: dc . b "E"
F: dc . b "F"
G: dc . b "G"
H: dc . b "H"
```

It is similar with `Lo/HiRepeatable`. Here one bit is reserved for one key in `HiKeyMap` and `LoKeyMap`. The set bits indicate if the corresponding key should be repeated after it has been released (see input device). If, for example, this bit = 0 for the <Return> key, the <Return> key does not repeat.

How can you use a new keymap over the console device? The keymap structure must be filled by `Console_AskKeyMap()` with the values of the console window keymap structure.

```
struct KeyMap KeyMap;
   ...
   Console_AskKeyMap(ConsoleRead, &KeyMap);
```

Then change the corresponding pointer of the keymap structure and the command `CD_SETKEYMAP`.

```
   KeyMap.km_LoKeyMapTypes = (UBYTE*) &LoKeyMapTypes;
   KeyMap.km_LoKeyMap      = (ULONG*) &LoKeyMap;
   KeyMap.km_HiKeyMapTypes = (UBYTE*) &HiKeyMapTypes;
   KeyMap.km_HiKeyMap      = (ULONG*) &HiKeyMap;
   Console_SetKeyMap(ConsoleRead, &KeyMap);
```

Now the new keymap is installed. Here are the `Con_Support` routines that we have used above, add them to your Con_Support.c file:

```
/****************************************************************
*                      Console_AskKeyMap()      (Con_Support)*
*                                                              *
* Function: Fill KeyMap-Structure                              *
*--------------------------------------------------------------*
* Input - Parameter:                                           *
*                                                              *
* ConReq:   Device-Block                                       *
* KeyMap:   Pointer to KeyMap-Structure                        *
*--------------------------------------------------------------*
* Return Value:                                                *
*                                                              *
* FALSE: Error !!!                                             *
****************************************************************/

BOOL Console_AskKeyMap (ConReq,KeyMap)
struct IOStdReq        *ConReq;
struct KeyMap                *KeyMap;
{

   ConReq->io_Length = (sizeof(struct KeyMap));
   ConReq->io_Data   = (APTR)KeyMap;
   Do_Command (ConReq,(UWORD)CD_ASKKEYMAP);
   if (ConReq->io_Error != (BYTE)0) return (FALSE);
   return (TRUE);
}

/****************************************************************
*                      Console_SetKeyMap()      (Con_Support)*
*                                                              *
* Function: Install Console-KeyMap                             *
```

```
*-------------------------------------------------------------*
* Input - Parameter:                                          *
*                                                             *
* ConReq:   Device-Block                                      *
* KeyMap:   Pointer to KeyMap-Structure                       *
*-------------------------------------------------------------*
* Rerun value:                                                *
*                                                             *
* FALSE: Error !!!                                            *
***************************************************************/

BOOL Console_SetKeyMap (ConReq,KeyMap)
struct IOStdReq       *ConReq;
struct KeyMap                 *KeyMap;
{

   ConReq->io_Length = (sizeof(struct KeyMap));
   ConReq->io_Data   = (APTR)KeyMap;
   Do_Command (ConReq,(UWORD)CD_SETKEYMAP);
   if (ConReq->io_Error != (BYTE)0) return (FALSE);
   return (TRUE);
}
```

You also have the option of bypassing the console device and assigning a different keymap to the console window. This can be done through `CD_ASKDEFAULTKEYMAP` and `CD_SETDEFAULTKEYMAP`. When you use these two commands to install a new keymap, the next time you invoke the `Open_A_Device ("console.device", 0L, &ConsoleRead, 0L, 0L);` command enables the keymap. These commands are used by the `SetMAP` CLI command. The `SetMAP` command changes the ConUnit structure of the current CLI window.

```
/**************************************************************
*                   Console_AskDelfaultKeyMap()   (Con_Support)*
*                                                             *
* Function: Fill KeyMap-Structure                             *
*-------------------------------------------------------------*
* Input - Parameter:                                          *
*                                                             *
* ConReq:   Device-Block                                      *
* KeyMap:   pointer to KeyMap-Structure                       *
*-------------------------------------------------------------*
* Retrun value:                                               *
*                                                             *
* FALSE: Error !!!                                            *
***************************************************************/

BOOL Console_AskDefaultKeyMap (ConReq,KeyMap)
struct IOStdReq               *ConReq;
struct KeyMap                         *KeyMap;
{

   ConReq->io_Length = (sizeof(struct KeyMap));
   ConReq->io_Data   = (APTR)KeyMap;
   Do_Command (ConReq,(UWORD)CD_ASKDEFAULTKEYMAP);
   if (ConReq->io_Error != (BYTE)0) return (FALSE);
   return (TRUE);
}
```

```
/*****************************************************************
*                 Console_SetDefaultKeyMap()      (Con_Support)*
*                                                              *
* Function: Install Console-Default-KeyMap                     *
*--------------------------------------------------------------*
* Input - Parameter:                                           *
*                                                              *
* ConReq:   Device-Block                                       *
* KeyMap:   Pointer to KeyMap-Structure                        *
*--------------------------------------------------------------*
* Retrun value:                                                *
*                                                              *
* FALSE: Error !!!                                             *
*****************************************************************/

BOOL Console_SetDefaultKeyMap (ConReq,KeyMap)
struct IOStdReq              *ConReq;
struct KeyMap                       *KeyMap;
{

   ConReq->io_Length = (sizeof(struct KeyMap));
   ConReq->io_Data   = (APTR)KeyMap;
   Do_Command (ConReq,(UWORD)CD_SETDEFAULTKEYMAP);
   if (ConReq->io_Error != (BYTE)0) return (FALSE);
   return (TRUE);
}
```

4.9.2 Console internals

After `Open_A_Device` the `ConsoleRead->io_Unit` points to a `ConUnit` structure. This structure contains all of the important variables needed for using the console:

```
Offset    Structure
------    --------            struct ConUnit
          {
0   0x00  struct MsgPort  cu_MP;  /* message port for sending */
                                       /* and receiving       */
34  0x22     struct Window  *cu_Window;    /* Console Window */
38  0x26     WORD           cu_XCP;
40  0x28     WORD           cu_YCP;      /* character position */
42  0x2a     WORD           cu_XMax;
44  0x2c     WORD           cu_YMax;      /* maximum character */
                                           /* position        */
46  0x2e     WORD           cu_XRSize;
48  0x30     WORD           cu_YRSize;         /* character size */
50  0x32     WORD           cu_XROrigin;
52  0x34     WORD           cu_YROrigin;      /* start of the    */
                                              /* text region     */
54  0x36     WORD           cu_XRExtant;  /* maximum size      */
56  0x38     WORD           cu_YRExtant;  /* of the text region */
58  0x3a     WORD           cu_XMinShrink;        /* smaller     */
60  0x3c     WORD           cu_YMinShrink;          /* unrelated */
                                                  /* region by   */
                                               /* Window Resize */
62  0x3e     WORD           cu_XCCP;
```

```
64  0x40     WORD            cu_YCCP;          /* Cursor Position */
66  0x42     struct KeyMap    cu_KeyMapStruct;          /* KeyMap */
98  0x62     UWORD           cu_TabStops[80];    /* Tab Positions */
/* see RastPort Structure: */
178 0xb2    BYTE                cu_Mask;
179 0xb3    BYTE                cu_FgPen;
180 0xb4    BYTE                cu_BgPen;
181 0xb5    BYTE                cu_AOLPen;
182 0xb6    BYTE                cu_DrawMode;
183 0xb7    BYTE                cu_AreaPtSz;
184 0xb8    APTR                cu_AreaPtrn;  /* Cursor Pattern */
188 0xbc    UBYTE               cu_Minterms[8];
196 0xc4    struct TextFont *cu_Font;
200 0xc8    UBYTE               cu_AlgoStyle;
201 0xc9    UBYTE               cu_TxFlags;
202 0xca    UWORD               cu_TxHeight;
204 0xcc    UWORD               cu_Tx_Width;
206 0xce    UWORD               cu_TxBaseline;
208 0xd0    UWORD               cu_TxSpacing;
210 0xd2    UBYTE            cu_Modes[3];   /* memory for modes */
                                            /* and RAW EVENTS   */
                                        /* (respectively 1 Bit) */
213 0xd5    UBYTE               cu_RawEvents[3];
216 0xd8  }                  /* defined in "devices/conunit.h" */
```

4.9.2.1 Console functions

The console device includes functions which can be accessed through offsets. The `ConsoleDevice = ConsoleRead->io_Device` represents the basis address for these functions.

```
Offset    Command
------    -------0x2a    CDInputHandler(&InputEvent)
                           A0
-0x30     Actual = RawKeyConvert(&InputEvent, Buffer, Length,
KeyMap)
          D0                          A0         , A1    , D1    , A2
```

`CDInputHandler()` sends the event in the input device to the current console window. For example, you can initialize an input event structure and send it to the console device using `CDInputHandler`. The input event's reaction can then be displayed by the console device. `CDInputHandler` is similar to `CMD_WRITE`, except that `CDInputHandler` displays event structures instead of strings.

`CDInputHandler()` calls `RawKeyConvert()`. This command translates the input event into a string that begins at the `Buffer` parameter, and has a maximal length of `Length`. This specifies the keymap intended for conversion. `Actual` contains the number of characters which comprise the created string. If `Actual` contains the value -1, the buffer wasn't large enough to hold the string.

A console function can be called using the following syntax:

```
move.l  _ConsoleDevice, a6
;initialize parameter
jsr     -$2a(a6)
```

4.9.2.2 More key codes

Take another look at the keymap illustration that appeared earlier in this book. You'll notice that beside some undefined keyboard codes there are also two reserved keyboard codes ($2b and $30). These keyboard codes are intended for foreign characters.

In addition to the keyboard codes from 0x00 through 0x67 there are more codes that cannot be controlled through a keymap:

```
0x68    left mouse button
0x69    right mouse button
0x6a    middle mouse button
```

These three key codes are never sent from the console device during normal operation. First, if you turn on the RAW mode for the mouse keypress with "<CSI>2{", you can add these key codes through the control string that was received.

0x80-0xe7 The key assigned a code ranging from 0x00 to 0x67 was released (e.g., 0x80 for key 0x00). This code can only be received when the flag `KCF_DOWNUP` (0x80) is set for the key in `KeyMapTypes`.

0xf9 The keyboard code last sent from the keyboard was incorrect.

0xfa The internal keyboard buffer (10 characters) is full.

0xfb Keyboard catastrophe - fatal error.

0xfd Keyboard power-up (keys pressed during booting). This was sent from 0xfd and 0xfe. (for example: 0xfd, 0x03, 0x04, 0xfe).

0xfe Keyboard power-up ended (keyboard initialized and all keys pressed in the meantime are sent to the system).

0xff The mouse was moved (no button pressed).

4.10 The clipboard device

You have probably worked with the block movement operations included in a word processor or text editor. Block operations require some memory management. If you've ever thought about adding block operations to your own programs, you may have changed your mind when you thought about how complicated this memory management can be.

The clipboard device offers a simple method of implementing block commands. You allocate the block into which you want the clipboard device to write. The clipboard device reserves this block until you access it further or declare it as invalid.

The clipboard device can be opened as follows:

```
struct IOClipReq *ClipReq = 0L;
#define CLIP_LEN (ULONG) sizeof(struct IOClipReq)
...
   Open_A_Device("clipboard.device", Unit, &ClipReq, 0L,
CLIP_LEN);
...
```

The clipboard device can only handle one unit at a time. Therefore, you must open the clipboard device with different unit numbers, if you wish to access more than one unit. The device block layout below will help you understand how this works:

```
Offset    Structure
------    --------          struct IOClipReq
          {
0   0x00   struct Message  io_Message;
20  0x14   struct Device  *io_Device;
24  0x18   struct Unit    *io_Unit;     /* which unit? */
28  0x1c   UWORD           io_Command;
30  0x1e   UBYTE           io_Flags;
31  0x1f   UBYTE           io_Error;
32  0x20   ULONG           io_Actual;          /* number of */
                                        /*transferred bytes */
36  0x24   ULONG           io_Length;  /* number of bytes to*/
                                              /*  transfer */
40  0x28   SPTR            io_Data;  /* data (Stringpointer)*/
44  0x2c   ULONG           io_Offset; /* Offset inside unit */
48  0x30   LONG         io_ClipID; /* identification number */
                                             /* of the clip */
52  0x34  } /* defined in "devices/clipboard.h" */
```

The variables `io_Message`, `io_Device`, and so on may be familiar to you from the previous section.

The variables `io_Offset` and `io_ClipID` are of interest. The clipboard device must reserve memory locations before it can save data to memory. The `io_Offset` variable helps determine the position at which the data was last read/written. `io_Offset` contains the byte offset inside the clipboard device that gives the last read/write position. This variable is similar to the file position used by DOS to determine the location of the file. The `io_ClipID` variable contains the number of blocks that was already written to the clipboard device, then deleted from the clipboard device.

The following routine writes data to the clipboard using the `CMD_WRITE` command:

```
/**********************************************************
*                    Clip_Write()          (Clip_Support)*
*                                                        *
* Function: Write data in the ClipBoard-Device n         *
*--------------------------------------------------------*
* Input - Parameter:                                     *
*                                                        *
* ClipReq: Device-Block                                  *
* Data:    Data to be written                            *
* Len:     Number of bytes to write                      *
* FirstTime: TRUE  => first write command                *
*            FALSE => write command of a sequence        *
*--------------------------------------------------------*
* Return value:                                          *
*                                                        *
* Number of data written                                 *
**********************************************************/

ULONG Clip_Write (ClipReq,Data,Len,FirstTime)
struct IOClipReq *ClipReq;
APTR                         Data;
LONG                              Len;
BOOL                                  FirstTime;
{
   if (FirstTime==TRUE)
      ClipReq->io_Offset = 0l;

   ClipReq->io_Data    = (STRPTR) Data;
   ClipReq->io_Length  = Len;
   Do_Command (ClipReq, (UWORD) CMD_WRITE);
   return (ClipReq->io_Actual);
}
```

You must set the variables `io_Offset` and `io_ClipID` to zero on the first write access. This prevents the clipboard device from acting through another device. To inform the clipboard device that all of the data was written, a `CMD_UPDATE` command is sent after the `CMD_WRITE` command. You can also write a larger block bit by bit to the clipboard device.

The clipboard device now contains a block. This block can consist of text, graphics or other data. If not enough memory was allocated to the clipboard block, the clipboard device writes your block to disk, placing it in the directory "SYS:devs/clipboards". The filename is the unit number, notated as a decimal string (e.g., 0).

The following routine reads data from a clipboard device block using the CMD_READ command:

```
/**********************************************************
*                    Clip_Read()            (Clip_Support)*
*                                                          *
* Funktion: Read data from ClipBoard-Device                *
*----------------------------------------------------------*
* Input - Parameter:                                       *
*                                                          *
* ClipReq: Device-Block                                    *
* Data:    Data buffer                                     *
* Len:     Number of bytes to read                         *
* FirstTime: TRUE  => first read command                   *
*            FALSE => read command of a sequence           *
*----------------------------------------------------------*
* Return value:                                            *
*                                                          *
* Number of the data that was read (contains errors!)      *
***********************************************************/

ULONG Clip_Read  (ClipReq,Data,Len,FirstTime)
struct IOClipReq *ClipReq;
APTR                         Data;
LONG                              Len;
BOOL                                  FirstTime;
{
   if (FirstTime=TRUE)
      ClipReq->io_Offset = 0l;

   ClipReq->io_Data   = (STRPTR) Data;
   ClipReq->io_Length = Len;
   Do_Command (ClipReq,(UWORD) CMD_READ);
   return (ClipReq->io_Actual);
}
```

The `io_Offset` and `io_ClipID` variables must be set to zero on the first read access.

There is a bug in the clipboard device which we must mention here. A zero is usually placed in `io_Actual` to indicate that all of the data has been read. Unfortunately, the clipboard device always takes the value in `io_Length` and places it in `io_Actual`. You must also keep the number of bytes written in the footer yourself so that none of the data is lost. In addition, you must read the data once with a value of zero in `io_Length` after all of the data has been read. This serves to end a read sequence for the clipboard device when the value zero is returned in `io_Actual`. Because the value from `io_Length` is always transferred into `io_Actual` when reading, you must read zero byte data.

When you no longer need the block, you must execute the CMD_CLEAR command. This clears the data in the clipboard device and increments the ClipID counter by one. For write and read accesses you should only clear the io_Offset array. Do not write to io_ClipID, for reasons which we'll explain in a moment.

Converting a large block to another format from the clipboard device (e.g., conversion to IFF format) can take a lot of time. You can save time by declaring a Clip (naming the transfer of data blocks in and out of the clipboard device). First you specify the address of a message port to the io_Data pointer. Through this message port you get a Satisfy message if the data available for use is needed by both parts. This also sends a CBD_POST command. The Satisfy message looks like the following structure:

```
Offset    Structure
------    --------          struct SatisfyMsg
          {
0  0x00      struct Message sm_Message;
20 0x14      UWORD          sm_Unit; /* from which unit */
22 0x16      LONG           sm_ClipID;
26 0x1a   }
```

You can then test for a received message using Message = GetMessage (ownPort). A received message is indicated by (Message !=0). If not, your program can continue with other tasks. If a message has arrived, you must write the data into the clipboard device as described above. Remember that the clip announced with POST is not needed in some cases. Meanwhile other blocks can be written to and read from the clipboard device. This naturally changes the io_ClipID variables.

When you want to get a POST command after receiving the Satisfy message through a CMD_WRITE command, read the io_ClipID of the current read command with CBD_CURRENTREADID. If this value is larger than the io_ClipID variable of the device block, with which the CBD_POST command is executed, your data is not needed.

If you want to test whether you should execute a previously sent POST command before leaving the program, just read the io_ClipID. Compare its contents with the io_ClipID variable of the POST device block. If the value returned from CURRENTWRITEID in io_ClipID is larger than the ClipID variable of the POST device block, the other CMD_WRITE commands are executed in the meantime and the announced data transfer does not need to be executed.

These two routines are combined to form the Clip_Support.c file, remember to insert the following include files, exec/types.h, exec/memory.h, exec/io.h, and devices/clipboard.h.

4.11 The audio device

You've probably heard about the fantastic sound capabilities of the Amiga. Programming sound requires the use of the audio device. This device allows you to send any waveform through the sound channels, at any volume and of any duration.

The audio device can be opened using the following:

```
#define AUDIO_LEN (ULONG) sizeof(struct IOAudio)
struct IOAudio *Audio_Request=0L;
...
  Open_A_Device("audio.device", 0L, Audio_Request, 0L,
AUDIO_LEN);
...
```

The `IOAudio` device block through which the command is given and developed looks like the following:

```
Offsets    struct IOAudio
-------    {/* defined in "devices/audio.h" */
0   0x00      struct IORequest  ioa_Request; /* IORequest to
                                                 begin */
32  0x20      WORD              ioa_AllocKey;
34  0x22      UBYTE            *ioa_Data;     /* Data pointer */
38  0x26      ULONG             ioa_Length;  /*size data field*/
42  0x2a      UWORD             ioa_Period;  /* Frequency */
44  0x2c      UWORD             ioa_Volume;  /* volume */
46  0x2e      UWORD             ioa_Cycles;  /* cycles */
48  0x30      struct Message    ioa_WriteMessage;
62  0x3d   }
```

The audio device supports the following commands:

ADCMD_ALLOCATE	(32)	allocate sound channel
ADCMD_FINISH	(11)	end sound output
ADCMD_FREE	(9)	unlock sound channel
ADCMD_LOCK	(13)	clean up before channel "stolen"
ADCMD_PERVOL	(12)	adjust period and volume
ADCMD_SETPREC	(10)	change channel precedence
ADCMD_WAITCYCLE	(14)	wait for end of cycle
CMD_FLUSH	(8)	clear all write commands
CMD_READ	(2)	find current `writeIO` block
CMD_RESET	(1)	reset audio hardware registers
CMD_START	(7)	start output
CMD_STOP	(6)	stop output
CMD_WRITE	(3)	initialize sound output

The audio device supports the following flags:

ADIOF_PERVOL	(16)	set period and volume using ADCMD_ALLOCATE
ADIOF_SYNCCYCLE	(32)	synchronize action with cycles
ADIOF_NOWAIT	(64)	don't wait for ADCMD_ALLOCATE

The audio device supports the following errors:

ADIOERR_NOALLOCATION	(-10)	AllocKey not understood
ADIOERR_ALLOCFAILED	(-11)	channel allocation failed
ADIOERR_CHANNELSTOLEN	(-12)	channel stolen by another user

4.11.1 Allocating audio channels

The audio channels must be allocated for sound transmission. The audio device has two methods of allocating the audio channels through which the sound can be sent.

Audio channel allocation: Method one

OpenDevice() provides the first method. First the IOAudio structure (the I/O block of the audio device) must be initialized. Initialization through CreateExtIO() alone isn't enough here—the IOAudio structure must be opened as well. Next an attempt is made to allocate the channels by calling OpenDevice() (Exec function). This audio device block requires the address of your channel allocation map, your audio channel reservation mask:

```
struct IOAudio *AudioReq;
char Channel_Map[...] = {...};
...
   AudioReq = (struct IOAudio *) GetDeviceBlock(AUDIO_LEN);
   AudioReq->ioa_Data    = Channel_Map;
   AudioReq->ioa_Length = sizeof(Channel_Map);
   Open_A_Device("audio.device", 0L, AudioReq, 0L, AUDIO_LEN);
```

A typical channel allocation map consists of up to 16 bytes, with which you can determine channel allocation. The four lowest bits (the low nibble) test for one of each of the bytes of the channels to be allocated. If you want to send your waveform to a left and a right sound channel, your allocation map would look like the following:

```
#define Left_Channel_0  1     /* Bit for first left channel */
#define Right_Channel_1 2    /* Bit for first right channel */
#define Right_Channel_2 4   /* Bit for second right channel */
#define Letf_Channel_3  8    /* Bit for second left channel */
UBYTE Channel_Map[] = {Left_Channel_0 | Right_Channel_1,
                       Left_Channel_0 | Right_Channel_2,
                       Left_Channel_3 | Right_Channel_1,
                       Left_Channel_3 | Right_Channel_2};
```

The `channel_map` array contains four entries (`sizeof (Channel_Map) == 4`) that test a right and a left sound channel for allocation. This array appeared in the previously initialized `IOAudio` structure (see the `ioa_data` pointer). The number of entries in this array is given through the `ioa_Length` pointer. In this particular case the length is four bytes. The following sequence is needed for the allocation of the channels using `OpenDevice()`:

```
#define Left_Channel_0  1
#define Right_Channel_1 2
#define Right_Channel_2 4
#define Letf_Channel_3  8
#define Precedence      -40
#define AUDIO_LEN sizeof(struct IOAudio)
UBYTE Channel_Map[] = {Left_Channel_0 | Right_Channel_1,
                       Left_Channel_0 | Right_Channel_2,
                       Left_Channel_3 | Right_Channel_1,
                       Left_Channel_3 | Right_Channel_2};
struct IOAudio *AudioReq;
...
   AudioReq = (struct IOAudio *) GetDeviceBlock(AUDIO_LEN);
   AudioReq->ioa_Data    = (UBYTE*)Channel_Map;
   AudioReq->ioa_Length  = (ULONG)sizeof(Channel_Map); /* 4 */
   AudioReq->ioa_Request.io_Message.mn_Node.ln_pri = Precedence;
   Open_A_Device("audio.device", 0L, &AudioReq, 0L, 0L);
```

Since we used `GetDeviceBlock` to take the audio block ourselves, this does not have to be done through `Open_A_Device()`. This is why we assign `Open_A_Device()` a value of zero as the size of the audio device block.

Now the supplied audio channels are ready for use (in case `OurSounds->ioa_Request.io_Error == 0`). You may have been wondering what the following variable does in this device:

```
AudioReq->ioa_Request.io_Message.mn_Node.ln_Pri
```

The above line controls the precedence of your sounds. The higher the precedence number, the less chance of your channel being "stolen" by another user. Stealing refers to another program allocating a sound channel you already have allocated. If your channel has a high precedence, a program trying to allocate your sound channel will be rejected. If the other program has a precedence number higher than yours, your channel must be released as soon as possible. Commodore-

Amiga recommends the following precedence numbers for certain types of sounds:

Precedence number:	Sound type
128	This precedence number is for the lazy programmer. When your sounds run at a level of 127, no one can access your allocated channels. You also don't have to free the channels before leaving the program. Use this number only when absolutely necessary!
90 - 100	Emergency sounds. These sounds occur when a problem occurs in an application (e.g., the application cannot access a library).
80 - 90	Announcements (e.g., the <Ctrl><G> bell).
75	Narrator device data (i.e., speech).
50 - 70	Informational sounds or sonic cues.
-50 - +50	Music program data.
-70 - 0	Sound effects (e.g., explosions).
-100 - -80	Background music and ambient sounds.
-128	Total silence.

Audio channel allocation: Method two

The second method works through an audio device command. We have placed this command (ADCMD_ALLOCATE) in an audio device support function which looks like the following:

```
/*************************************************************
*                       Audio_Allocate()        (Audio_Support)*
*                                                            *
* Function: Allocate sound channels or audio device block    *
*           Prepare for allocation (OpenDevice)              *
*------------------------------------------------------------*
* Input - Parameter:                                         *
* Audio_Device_Block: Device-Block for allocation            *
* Channel_Map:        Channel-allocations-mask               *
* Size:               Size of the allocations-mask (BYTES)   *
* Precedence:         Sound precedence (-127 - 128)          *
* Wait:               Wait for the desired channel to open?  *
*------------------------------------------------------------*
* Return value:                                              *
* Error in command execution                                 *
*************************************************************/
BYTE Audio_Allocate
(Audio_Device_Block,Channel_Map,Size,Precedence,Wait)
struct IOAudio       *Audio_Device_Block;
```

```
UBYTE                                    *Channel_Map;
ULONG                                            Size;
BYTE
Precedence;
BOOL
Wait;
{
   Audio_Device_Block->ioa_Data                = Channel_Map;
   Audio_Device_Block->ioa_Length              = Size;

   Audio_Device_Block->ioa_Request.
                    io_Message.mn_Node.ln_Pri     = Precedence;

   if (!Wait) /* wait until channel is free? */
      Audio_Device_Block->ioa_Request.io_Flags   |= (UBYTE)
ADIOF_NOWAIT;
   if (Audio_Device_Block->ioa_Request.io_Device != 0L)
   {
      Audio_Device_Block->ioa_Request.io_Command  = (UWORD)
ADCMD_ALLOCATE;
      if (Wait) DoIO (Audio_Device_Block);
      else
      {
         SendIO (Audio_Device_Block);
         if (CheckIO(Audio_Device_Block) == 0) return
(ADIOERR_ALLOCFAILED);
      }
      return (Audio_Device_Block->ioa_Request.io_Error);
   }
   return (0x00);
}
```

This routine sets the `io_Data` pointer (`Channel_Map`) and the `ioa_Length` variable (size of the `Channel_Map`) as well as the user assigned sound precedence. The `NOWAIT` flag (Wait = FALSE) is also set to the user-assigned value. To allocate channels contained by a higher precedence request while the `NOWAIT` flag is set, give `OpenDevice()` and `ADCMD_ALLOCATE` the error `ADIOERR_ALLOCFAILED` as the return value or error flag (`AudioReq->ioa_Request.io_Error`).

The unset `NOWAIT` flag waits until the request containing our channels releases them. A small problem can occur when you use the `NOWAIT` flag in conjunction with `ADCMD_ALLOCATE`, it cannot be used for channel allocation with `OpenDevice()`. You might think that by returning the set `NOWAIT` flag `ADCMD_ALLOCATE` to the called program, the channels can be allocated. This is done in case the supplied channels are inaccessible, but unfortunately they wait for `ADCMD_ALLOCATE`.

We got around this error with the help of the `SendIO()` and `CheckIO()` commands. If channels should be released without delay, we send a asynchronous request (`SendIO()`) and check it immediately to see if the sent command is executing, or if it was already processed. If `CheckIO()` returns the value 0, that is one character for us that the channels contain from others. Because we do not want to wait for the channels to be released, the routine exits with the error ADIOERR_ALLOCFAILED.

If the channels could not be allocated, you get the bit combination of the allocated channels in *Audio_Device_Block->ioa_Request.io_unit. It is important to know which channels were actually allocated (more on this later). With multiple statements of the channels to be allocated in the Channel_Map (up to 16), ADCMD_ALLOCATE searches for the combination that requires the least wait time.

If in the meantime some other program releases its sound channels, ADCMD_ALLOCATE or OpenDevice() checks to see if some successful allocation can be executed. This check results only when the system is instructed to wait for released channels. Should the allocation fail, the error variable of the Audio_Device_Block sets the ALLOC_FAILED flag:

```
/*****************************************************************
*                       Audio_NoAlloc()         (Audio_Support)*
*                                                               *
* Function: Allocation succesful?                               *
*---------------------------------------------------------------*
* Input - Parameter:                                            *
*                                                               *
* Audio_Device_Block: Device-Bock to be tested                  *
*---------------------------------------------------------------*
* Return value:                                                 *
*                                                               *
* TRUE:  No allocation                                          *
* FALSE: Allocation successful                                  *
*****************************************************************/
BOOL Audio_NoAlloc (Audio_Device_Block)
struct IOAudio     *Audio_Device_Block;
{
   if ((Audio_Device_Block->ioa_Request.io_Error &
ADIOERR_NOALLOCATION) ==  ADIOERR_NOALLOCATION)
      return (TRUE);
   else
      return (FALSE);
}
```

The AllocKey variable contains information about how many users exist on the audio device when the computer is turned on (or reset). When this AllocKey does not agree with the internally stored value when an audio device command is executed, each audio device command sends a NOALLOCATION error.

To avoid generating this error when you want to use a copy of the original device block, this AllocKey is also copied when copying the audio device block. Before we get to that, let's take a closer look at the allocation. You can save work when allocating by using

OpenDevice(). When the ADIOF_PERVOL flag is set, the period (frequency) and volume are also set. You can also do this for each period with the help of an audio command.

4.11.2 Audio period and volume

The following routine demonstrates volume setting for the audio device:

```
/***************************************************************
*                      Audio_Pervol()        (Audio_Support)*
* Function: Set volume and frequency                           *
*-------------------------------------------------------------*
* Input - Parameter:                                           *
*                                                              *
* Audio_Device_Block: Audio-Device-Block should be set for     *
*                     volume and frequency                     *
* Period:             Sample frequency                         *
* Volume:             Volume                                   *
*-------------------------------------------------------------*
* Return value:                                                *
*                                                              *
* Error encounter during commnad execution                     *
**************************************************************/
BYTE Audio_Pervol (Audio_Device_Block, Period, Volume)
struct IOAudio     *Audio_Device_Block;
UWORD                                    Period;
UWORD                                            Volume;
{
   Audio_Device_Block->ioa_Request.io_Flags    = (UBYTE)
ADIOF_SYNCCYCLE;
   Audio_Device_Block->ioa_Period              = Period;
   Audio_Device_Block->ioa_Volume              = Volume;

   if (Audio_Device_Block->ioa_Request.io_Device != 0L)
   {  /* setting of volume and frequency */
      Audio_Device_Block->ioa_Request.io_Command  = (UWORD)
ADCMD_PERVOL;
      DoIO (Audio_Device_Block);
      return (Audio_Device_Block->ioa_Request.io_Error);
   }

   Audio_Device_Block->ioa_Request.io_Flags |= ADIOF_PERVOL;
   /* ADIOF_PERVOL must be set for OpenDevice() */
   return (0x00);
}
```

This routine can tell whether the volume should be set before or after OpenDevice(). The io_Device pointer as well as all unset variables should equal zero before calling OpenDevice(), because the allocation of the audio device block (GetDeviceBlock()) has been executed through AllocMem() with the MEMF_PUBLIC and MEMF_CLEAR parameters.

Three options exist for setting the volume and period. The `ADIOF_PERVOL` flag, used in conjunction with `CMD_WRITE` (sound output) has the same function as when used in conjunction with `OpenDevice()`. Before we get to the sound output itself, let's take a closer look at the period and the volume.

We mentioned that the word period refers to the frequency of the sound. The period measurement requires a few simple calculations. For example, a period value of 20000 means more than just an output of 20000 bytes per second. You can calculate the period from two items:

1.) The number of bytes (samples) on which the waveform is based.

2.) The frequency at which the note(s) should be played.

Look at the following equation:

$$\text{Period} = \frac{1}{\text{Sampling rate} * 28 * 10^{-8}}$$

The sampling rate is constructed from the number of bytes to be played and the frequency at which these should be played.

Here's an example:. You want to play a waveform created from 40 bytes, and you want this waveform played at 440 cycles per second. This means that you want the waveform to sound at 440 Hz, or "A-440" as it's called in the music dictionaries. To get the sampling rate you do the following:

```
Sampling rate = 40 * 440 (40 bytes * 440 Hz)
```

The period is then calculated as follows:

$$\text{Period} = \frac{1}{40 * 440 * 28 * 10^{-8}} = 202.922 \approx 203$$

The following routine performs period calculation. We wanted to minimize floating point arithmetic, so we multiplied the above fraction by 10^8 (28*10^8seconds = 280 nanoseconds [the time in which one byte can be output]), and we get the following routine:

```
/****************************************************************
*                     Audio_Period()          (Audio_Support)*
*                                                              *
* Function: Calculate the number of bytes to play and          *
*           the frequency needed for it.                       *
*--------------------------------------------------------------*
* Input - Parameter:                                           *
*                                                              *
* Bytes: Number of bytes to play                               *
```

```
* Hz:    Frequency, at which the byte samples should be played *
*--------------------------------------------------------------*
* Return value:                                                *
*                                                              *
* Calculated period                                            *
****************************************************************/

UWORD Period (Bytes, Hz)
UWORD        Bytes, Hz;
{
   return ((UWORD) (1000000001 / (Bytes * Hz * (UWORD)28)));
}
```

Remember that this routine returns only variables of type UWORD. The value region ranges from 0 to 65535. If the period is larger than 65535, only the lower 16 bits of the calculation are used. The top 16 bits are truncated (see hardware register). In certain cases this can be done to very small values that lie outside of the allocated region. The period may not be smaller than 124. The frequency with which the bytes are displayed is found as follows:

```
    Period = 100000000L/(Bytes*Hz*28)
<=> Hz     = 100000000L/(Bytes*Period*28)
```

Assuming that our waveform only consists of one byte, this is then given as the highest frequency:

```
    Hz = 10000000L/(1*124*28) = 28800 s^-1;
```

We need a little more arithmetic to set the volume. The volume can range from zero (soft) to 64 (loud). The volume curve is linear.

Note: Set the period and volume to zero before using `CMD_WRITE` for the first time. If you don't do this, data displayed through `CMD_WRITE` may also play over the sound channels as blips.

4.11.3 Play it

Now we come to the frequently mentioned `CMD_WRITE` command. We have also included this in a short routine:

```
/***************************************************************
*                        Audio_Write()        (Audio_Support)*
*                                                              *
* Function: Output data through channel                        *
*--------------------------------------------------------------*
* Input - Parameter:                                           *
*                                                              *
* Audio_Device_Block: Device-Block                             *
* WaveForm:           Addess of the waveform array             *
* WaveLength:         Number of bytes to play                  *
```

```
* Cycles:            Number of waveform repetitions          *
* ComeBack:          Wait until end of CMD_WRITE (FALSE) ?    *
*-------------------------------------------------------------*
* Return value:                                               *
*                                                             *
* Error during command execution                              *
***************************************************************/

BYTE Audio_Write (Audio_Device_Block, WaveForm, WaveLength,
Cycles, ComeBack)
struct IOAudio   *Audio_Device_Block;
UBYTE                              *WaveForm;
ULONG                                          WaveLength;
UWORD                                                   Cycles;
BOOL                                                  ComeBack;
{
   Audio_Device_Block->ioa_Data               = WaveForm;
   Audio_Device_Block->ioa_Length             = WaveLength;
   Audio_Device_Block->ioa_Cycles             = Cycles;
   Audio_Device_Block->ioa_Request.io_Flags   = (UBYTE) 0;
   Audio_Device_Block->ioa_Request.io_Command = (UWORD)
CMD_WRITE;

   if (ComeBack) SendIO  (Audio_Device_Block);
   else          DoIO    (Audio_Device_Block);

   return (Audio_Device_Block->ioa_Request.io_Error);
}
```

We must provide the device block with which the device was opened, as with all of the `audio_support` routines.

`WaveForm` and `WaveLength` designate the waveform to be displayed. `WaveForm` contains the starting address of the byte array in which the waveform is located. `WaveLength` contains the number of bytes needed to describe the waveform, rather than the wave length of the resulting sound.

`Cycles` designates the repetition rate of a waveform. For example, if you want the given waveform to play only once, you must state the value 1 for `Cycles`. The higher the value for `Cycles`, the more times the waveform repeats.

When you want unlimited repetitions, enter the value zero for `Cycles`. Then your waveform plays for eternity, assuming that your channels aren't stolen. A change in the waveform can occur when the indicated location is displayed. Please remember that the sound data in the chip memory must be released. As you know, the address bus of the custom chips (Blitter, Paula, Agnus, Denise, Copper) contains only 19 address connections. This addresses the bottom section of memory (or the bottom 512K).

4.11.4 Additional audio device features

Now you can produce sound on your Amiga. You can allocate the necessary channels, determine the frequency and volume and make the data audible. Suppose you're working with your audio device and suddenly you don't hear anything anymore, or you don't hear what you expected to hear. The solution: You've been robbed of your channels.

What can you do? You can either leave the program, or reset the computer if you cannot leave the program. One possibility exists for consistently stealing channels, which involves the ADCMD_LOCK command:

```
#define AUDIO_LEN (ULONG) (sizeof(struct IOAudio))
VOID *GetDeviceBlock();
/***************************************************************
*                           Audio_Lock()         (Audio_Support)*
* Function: Protect channel before new access                  *
*--------------------------------------------------------------*
* Input - Parameter:                                           *
* Audio_Device_Block: Device-Block of the channels that        *
*                     should be protected.                     *
*--------------------------------------------------------------*
* Return value:                                                *
* Addess of the Lock                                           *
***************************************************************/

struct IOAudio *Audio_Lock (Audio_Device_Block)
struct IOAudio             *Audio_Device_Block;
{
   struct IOAudio *Lock;

   Lock = (struct IOAudio *)GetDeviceBlock(AUDIO_LEN);

   Audio_Copy (Audio_Device_Block,Lock);
   Lock->ioa_Request.io_Command = (UWORD)ADCMD_LOCK;

   SendIO (Lock);
   return (Lock);
}
```

ADCMD_LOCK is a command that executes first when the allocated channels are stolen. That means that as long as ADCMD_LOCK executes, everything is OK. If you want to use your audio channels, you should make sure you have access rights. The following routine should help you:

```
/****************************************************************
*                      Audio_Channel_Stolen()    (Audio_Support)*
*                                                               *
* Function: Was channel stolen?                                 *
*--------------------------------------------------------------*
```

```
* Input - Parameter:                                             *
*                                                                *
* Lock:  Device-Block of the Lock, that should be tested.        *
*----------------------------------------------------------------*
* Return value:                                                  *
*                                                                *
* TRUE:  Channel was stolen => Exit progra                       *
* FALSE: Channel is under your control                           *
******************************************************************/

BOOL Audio_Channel_Stolen (Lock)
struct IOAudio    *Lock;
{
   if (CheckIO (Lock) != 0)
      return (TRUE);
   else
      return (FALSE);
}
```

This short routine tests if the `LOCK` command has ended (`CheckIO() != 0`) or if the access right consists of the allocated channels. You may be wondering how you can output the data if you locked the audio channels. The `ADCMD_LOCK` command ends these channels when the channels are stolen. Processing the two device commands with the device block cannot be done at the same time.

A second device block must come into play. This is configured the same as other device blocks, using `GetDeviceBlock()`. Then it must ensure that you can also use the new device block. For this we have developed a copy function that also copies the `AllocKey` that is necessary for identifying the user of the audio device:

```
/*****************************************************************
*                       Audio_Copy()           (Audio_Support)*
* Function: Device-Block copy                                    *
*----------------------------------------------------------------*
* Input - Parameter:                                             *
* Old_Audio_Block: Original                                      *
* New_Audio_Block: Copy                                          *
******************************************************************/
VOID Audio_Copy (Old_Audio_Block, New_Audio_Block)
struct IOAudio   *Old_Audio_Block,*New_Audio_Block;
{
   New_Audio_Block->ioa_Request.io_Device =
      Old_Audio_Block->ioa_Request.io_Device;

   New_Audio_Block-d>ioa_Request.io_Unit =
      Old_Audio_Block->ioa_Request.io_Unit;

   New_Aiudio_Block->ioa_AllocKey =
      Old_Audio_Block->ioa_AllocKey;
}
```

When you want to leave the program, you cannot simply exit and leave the locked channels locked. In certain cases this would hinder every other sound output, because only channel applications with a lower precedence are executed. To prevent this from happening you must release the locked channels again, as shown in the following routine:

```
/****************************************************************
*                          Audio_Free()        (Audio_Support)*
*                                                               *
* Function: Remove protection from channels                    *
*--------------------------------------------------------------*
* Input - Parameter:                                            *
*                                                               *
* Lock: Device-Block of the freed lock                          *
****************************************************************/

VOID Audio_Free (Lock)
struct IOAudio  *Lock;
{
   Lock->ioa_Request.io_Command = (UWORD)ADCMD_FREE;
   DoIO (Lock);

   FreeDeviceBlock(Lock);
}
```

This routine frees the channels from your exclusive access and frees the previously allocated device block. Another, less elegant option exists for protecting channels from outside access: Set the precedence at 127. You can do this during allocation, or using the ADCMD_SETPREC command:

```
/****************************************************************
*                  Audio_SetPrecedence()       (Audio_Support)*
* Function: Chaneg precedence                                   *
*--------------------------------------------------------------*
* Input - Parameter:                                            *
*                                                               *
* Audio_Device_Block: Device-Block                              *
* Precedence:         New precedence                            *
****************************************************************/
VOID Audio_SetPrecedence (Audio_Device_Block,Precedence)
struct IOAudio          *Audio_Device_Block;
BYTE                                     Precedence;
{
   Audio_Device_Block->ioa_Request.io_Message.mn_Node.ln_Pri =
(BYTE)Precedence;
   Audio_Device_Block->ioa_Request.io_Command              =
(UWORD)ADCMD_SETPREC;
   DoIO (Audio_Device_Block);
}
```

Each call of ADCMD_SETPREC allows the check of ALLOCATE commands whether or not the new precedence is less than the old precedence. If this is the case, ALLOCATE can address the supplied channels for you.

When setting the precedence at 127 (the highest precedence possible) there is no chance for other ALLOCATE commands to get access to your channels. This occurs because the precedence of the channels that might steal them must be less than yours.

You now have the software means of writing your own sound programs. The following example is an example sound program. Combine the Audio_Support routines to form the Audio_Support.c file, don't forget the proper include files. Be careful of the order of the routines since Audio_Lock calls Audio_Copy.

```
/*************************************************************
*                      SoundEditor.c                         *
*                   (c) Bruno Jennrich                       *
*                      August 1988                           *
*************************************************************/
/*************************************************************
* Compile-Info:                                              *
* cc SoundEditor                                             *
* ln SoundEditor.o Audio_Support.o Devs_Support.o -lc        *
*************************************************************/
#include "exec/types.h"
#include "exec/memory.h"
#include "exec/devices.h"
#include "intuition/intuition.h"
#include "intuition/intuitionbase.h"
#include "graphics/gfxbase.h"
#include "graphics/gfxmacros.h"
#include "devices/audio.h"
#define ScreenHeight 200L          /* Editor-Screen */
#define ScreenWidth  3120L
#define ScreenDepth  2L
#define ScreenMiode   0L
#define ERROR         100
#define AUDIO_LEN     (ULONG) (sizeof (struct IOAudio))
VOID *OpenLibrary();
VOID *OpenScreen();
VOID *OpenWindow();
VOID *AllocMem();
VOID *GetDeviceBlock()
VOID *Audio_Lock();
struct Screen *Screen=0l;
struct Window *Window=0l;
struct NewScreen NewScreen;
struct NewWindow NewWindow;
struct GfxBase       *GfxBase=0l;
struct IntuitionBase *IntuitionBase=0l;
/* Audio-Device relevant structures */
#define Left_Channel_0  1
#define Right_Channel_1 2
#define Right_Channel_2 4
#define Left_Channel_3  8
#define SoundPrecedence (BYTE) -40
struct IOAudio *Left_Side        =0l,
               *Right_Side       =0l,
               *Left_Lock        =0l,
               *Right_Lock       =0l;
BYTE *WaveLeft  = 0l;    /* Wave form definition (signed) */
BYTE *WaveRight = 0l;
UBYTE    Left_Channels[]  = {Left_Channel_0,  Left_Channel_3};
UBYTE    Right_Channels[] = {Right_Channel_1, Right_Channel_2};
                                /* Channel_Map */
                                /* allocate right and left */
                                /* channels      */
#define CHANNELS_LEFT   (ULONG) sizeof ( Left_Channels)
```

```
#define CHANNELS_RIGHT  (ULONG) sizeof (Right_Channels)
                                /* Mask size */
/****************************************************************
*                              CloseIt()               (User)*
* Function: Close and free everything                         *
*-------------------------------------------------------------*
* Input - Parameter:                                          *
* String: Error-String                                        *
****************************************************************/
VOID CloseIt (String)
char        *String;
{
   UWORD i;
   UWORD *dff180 = (UWORD *)0xdff180;
   UWORD Error;
   if (strlen (String) > 0)
      for (i=0;i<0xffff;i++)  *dff180 = i;
   puts (String);
   if (Window != 0L)         CloseWindow  (Window);
   if (Screen != 0L)         CloseScreen  (Screen);
   if (GfxBase != 0L)        CloseLibrary (GfxBase);
   if (IntuitionBase != 0L) CloseLibrary (IntuitionBase);
   if (Left_Lock != 0L)      Audio_Free (Left_Lock);
   if (Right_Lock != 0L)     Audio_Free (Right_Lock);
   if (Left_Side != 0L)      Close_A_Device (Left_Side);
   if (Right_Side != 0L)     FreeDeviceBlock (Right_Side);
   if (WaveLeft  != 0L)      FreeMem(WaveLeft,
(ULONG)ScreenWidth);
   if (WaveRight != 0L)
FreeMem(WaveRight,(ULONG)ScreenWidth);

   exit (10);
}
/****************************************************************
*                        InstallScreenWindow()          (User)*
* Function: Editor Window and Screen initialization           *
****************************************************************/
VOID InstallScreenWindow ()
{
   NewScreen.LeftEdge      = 0;
   NewScreen.TopEdge       = 0;
   NewScreen.Width         = ScreenWidth;
   NewScreen.Height        = ScreenHeight;
   NewScreen.Depth         = ScreenDepth;
   NewScreen.DetailPen     = 1;
   NewScreen.BlockPen      = 0;
   NewScreen.ViewModes     = ScreenMode;
   NewScreen.Type          = CUSTOMSCREEN;
   NewScreen.Font          = (struct TextAttr *) 0l;
   NewScreen.DefaultTitle = (UBYTE *) " (c) Bruno Jennrich";
   NewScreen.Gadgets       = (struct Gadget *) 0l;
   NewScreen.CustomBitMap = (struct BitMap *) 0l;
   NewWindow.LeftEdge      = 0;
   NewWindow.TopEdge       = 0;
   NewWindow.Width         = ScreenWidth;
   NewWindow.Height        = ScreenHeight;
   NewWindow.DetailPen     = 1;
   NewWindow.BlockPen      = 0;
   NewWindow.IDCMPFlags    = 0;
   NewWindow.Flags         = BORDERLESS | ACTIVATE | RMBTRAP |
NOCAREREFRESH;
   NewWindow.FirstGadget  = (struct Gadget *) 0l;
```

```
   NewWindow.CheckMark    = (struct Image *)  0l;
   NewWindow.Title        = (UBYTE *) " Waveform-Editor";
   NewWindow.Screen       = (struct Screen *) 0l;
   NewWindow.BitMap       = (struct BitMap *) 0l;
   NewWindow.MinWidth     = 0;
   NewWindow.MaxWidth     = 0;
   NewWindow.MinHeight    = 0;
   NewWindow.MaxHeight    = 0;
   NewWindow.Type         = CUSTOMSCREEN;
}
/****************************************************************
*                         OpenScreenWindow()             (User)*
* Function: Screen and Window open                              *
****************************************************************/
VOID OpenScreenWindow ()
{
   InstallScreenWindow();

   Screen = (struct Screen *) OpenScreen (&NewScreen);
   if (Screen == 0l) CloseIt ("Couldn4t get Screen !");

   NewWindow.Screen = Screen;
   Window = (struct Window *) OpenWindow(&NewWindow);
   if (Window == 0l) CloseIt ("Couldn4t get Window !");
}
/****************************************************************
*                            CloseScreenWindow()         (User)*
* Function: Screen and Window close                             *
****************************************************************/
VOID CloseScreenWindow()
{
   CloseWindow (Window);
   CloseScreen (Screen);
}
/****************************************************************
*                            OpenLibs()                  (User)*
* Function: Open libraries                                      *
****************************************************************/
VOID OpenLibs ()
{
   GfxBase = (struct GfxBase *) OpenLibrary
("graphics.library",0l);
   if (GfxBase == 0l)      CloseIt ("Couldn4t get Grahpics !");
   IntuitionBase =
      (struct IntuitionBase *) OpenLibrary
("intuition.library",0l);
   if (IntuitionBase == 0l) CloseIt ("Couldn4t get Intuition
!");
}
/****************************************************************
*                            CloseLibs()                 (User)*
* Function: Libraries close                                     *
****************************************************************/
VOID CloseLibs ()
{
   CloseLibrary (GfxBase);
   CloseLibrary (IntuitionBase);
}
/****************************************************************
*                         The_Audio_Device()             (User)*
* Function: Use Audio-Device                                    *
****************************************************************/
```

```
The_Audio_Device()
{
   UWORD i,j;
   Open_A_Device ("audio.device",0L,&Left_Side,0L,AUDIO_LEN);
   Right_Side = (struct IOAudio*) GetDeviceBlock (AUDIO_LEN);
   Audio_Copy (Left_Side,Right_Side);
   /* allocate channel */
   if (Audio_Allocate (Right_Side,
                       Right_Channels,
                       CHANNELS_RIGHT,
                       SoundPrecedence,
                       FALSE) == ADIOERR_ALLOCFAILED)
      CloseIt ("Couldn4t get Right-Channel !");
   if (Audio_Allocate (Left_Side,
                       Left_Channels,
                       CHANNELS_LEFT,
                       SoundPrecedence,
                       FALSE) == ADIOERR_ALLOCFAILED)
      CloseIt ("Couldn4t get Left-Channel !");
   /* secure channels before foreign access */
   Left_Lock  = Audio_Lock (Left_Side);
   Right_Lock = Audio_Lock (Right_Side);
   if (Right_Lock == 0l) CloseIt ("Right_Lock failed !");
   if (Left_Lock  == 0l) CloseIt ("Left_Lock failed !");
   /* No sound output */
   Audio_Pervol (Left_Side, (UWORD)0,(UWORD)0);
   Audio_Pervol (Right_Side,(UWORD)0,(UWORD)0);
   /* Begin data output */
   Audio_Write (Left_Side,WaveLeft, ScreenWidth,
                                (UWORD) 0,(BOOL) TRUE);
   Audio_Write (Right_Side,WaveRight, ScreenWidth,
                                  (UWORD) 0,(BOOL) TRUE);
   /* Increase volume */
   Audio_Pervol (Right_Side,(UWORD)1500,(UWORD)64);
   Audio_Pervol (Left_Side,(UWORD)1500,(UWORD)64);
}
/****************************************************************
*                      Close_Audio_Device()              (User)*
* Function: Free channels and close Audio-Device                *
****************************************************************/
Close_Audio_Device()
{
   Audio_Free (Left_Lock);
   Audio_Free (Right_Lock);
   Close_A_Device (Left_Side);
   FreeDeviceBlock (Right_Side);
}
/****************************************************************
*                            Edit()                      (User)*
* Function: Wave form editor                                    *
****************************************************************/
Edit ()
{
   WORD  i;
   ULONG x,y;
   UBYTE *LeftMouse  = (UBYTE *) 0xbfe001;
   UWORD *RightMouse = (UWORD *) 0xdff016;
   Move (Window->RPort,0L,ScreenHeight/2L);
   Draw (Window->RPort,ScreenWidth,ScreenHeight/2L);
   for (i=0;i<ScreenWidth; i++)
   {
      WaveLeft [i] = 0;
```

```
         WaveRight[i] = 0;
      }
      SetDrMd (Window->RPort,(ULONG)COMPLEMENT);
      The_Audio_Device();
      while ((*RightMouse & 0x0400) == 0x0400)
      {
         if (Audio_Channel_Stolen (Left_Lock) ||
             Audio_Channel_Stolen (Right_Lock))
            CloseIt ("Channel stolen");
                          /* In case challels were stolen! */
         if ((*LeftMouse & 0x40) == 0)
         {
            x = Screen->MouseX;
            y = Screen->MouseY;
            if (WaveLeft[x] != (ScreenHeight/2-y))
            {
               Move (Window->RPort,x,ScreenHeight/2);
               Draw (Window->RPort,x,ScreenHeight/2-WaveLeft[x]);
               WaveLeft[x]  = (ScreenHeight/2-y);
               WaveRight[x] = WaveLeft[x];
               Move (Window->RPort,x,ScreenHeight/2);
               Draw (Window->RPort,x,ScreenHeight/2-WaveLeft[x]);
            }
         }
      }
      Close_Audio_Device();
   }
/****************************************************************
*                              main()                     (User)*
****************************************************************/
main()
{
   WaveLeft  = (BYTE*) AllocMem (ScreenWidth, (ULONG)
MEMF_CHIP|MEMF_CLEAR);
   WaveRight = (BYTE*) AllocMem (ScreenWidth, (ULONG)
MEMF_CHIP|MEMF_CLEAR);
   if ((WaveLeft == 0l) || (WaveRight == 0l))
      CloseIt ("No Wave Buffer !");
   OpenLibs();
   OpenScreenWindow();
   Edit();
   CloseScreenWindow();
   CloseLibs();
   FreeMem (WaveLeft,  ScreenWidth);
   FreeMem (WaveRight, ScreenWidth);
}
```

Double buffering

The next program presents a novel method of sound generation. For example, if you want to continuously play a sound, you know that your Amiga simply hasn't enough memory for playing 30 minutes of sampled sounds. Let's say you need a constant stream of background sound. You can play multiple samples repeatedly, using CMD_WRITES. For example:

```
Audio_Write (/* Sound 1 */);
Audio_Write (/* Sound 2 */);
Audio_Write (/* Sound 3 */);
...
```

This has a disadvantage: You can hear blips between the end of the previous sound and the start of the next sound. This is because the audio DMA channels pause long enough to look for the next file. This can be heard in the form of noise between the end of the old CMD_WRITE and the beginning of the new one.

You can create a double buffer for holding data. Double buffers are most often used in graphic programming. When you employ the technique of double buffering, these disturbing noises are no longer heard. The sound output results in the following scheme:

```
Play start sound
Loop:
      calculate new sound      (or load this from diskette)
      send write command for the new sound
      wait until the end of the old sound
                       (here the new sound is played)
      calculate another sound
      send write command for another sound
      wait for the old sound
repeat loop
```

Write commands and copies from the device blocks are listed internally and processed according to the order. For this reason the CMD_WRITE command does not break out for the same channels. Here's the double buffer program:

```
/****************************************************************
*                          Double.c                             *
*                     (c) Bruno Jennrich                        *
*                        August 1988                            *
* Compile-Info:                                                 *
* cc Double                                                     *
* ln Double.o Audio_Support.o Devs_Support.o -lc                *
****************************************************************/
#include "exec/types.h"
#include "exec/memory.h"
#include "exec/devices.h"
#include "devices/audio.h"
#define AUDIO_LEN     (ULONG) (sizeof (struct IOAudio))
VOID *OpenLibrary();
VOID *AllocMem();
VOID *GetDeviceBlock();
VOID *Audio_Lock();
/* Audio-Device relevant structures */
#define Left_Channel_0  1
#define Right_Channel_1 2
#define Right_Channel_2 4
#define Left_Channel_3  8
#define SoundPrecedence (BYTE) -40
#define WaveLength1 3201
#define WaveLength2 6001
struct IOAudio *FirstPlay        =01,
               *SecondPlay       =01,
               *FirstLock        =01,
               *SecondLock       =01;
char *FirstWave  = 01;    /* Wave form definition (signed) */
```

```
char *SecondWave = 0l;
UBYTE   Channels[]  = {Left_Channel_0, Left_Channel_3};
                               /* Channel_Map */
                               /* link */
                               /* allocate channel  */
#define CHANNEL_SIZE   (ULONG) sizeof (Channels)
                               /* Mask size */
/*************************************************************
*                           CloseIt()               (User)*
* Function: Close and free everything                       *
*-----------------------------------------------------------*
* Input - Parameter:                                        *
* String: Error-String                                      *
*************************************************************/
VOID CloseIt (String)
char        *String;
{
   UWORD i;
   UWORD *dff180 = (UWORD *)0xdff180;
   UWORD Error;
   if (strlen (String) > 0)
      for (i=0;i<0xffff;i++)  *dff180 = i;
   puts (String);
   if (FirstLock != 0)      Audio_Free (FirstLock);
   if (SecondLock != 0)     Audio_Free (SecondLock);
   if (SecondPlay != 0L)    FreeDeviceBlock (SecondPlay);
   if (FirstPlay != 0L)     Close_A_Device (FirstPlay);
   if (FirstWave != 0)      FreeMem (FirstWave,WaveLength1);
   if (SecondWave != 0)     FreeMem (SecondWave,WaveLength2);
   exit (10);
}
/*************************************************************
*                     The_Audio_Device()            (User)*
*                                                           *
* Funktion: Use Audio-Device                                *
*************************************************************/
The_Audio_Device()
{
   UWORD i,j;
   UBYTE *bfe001 = (UBYTE*) 0xbfe001;
   FirstPlay  = (struct IOAudio *) GetDeviceBlock (AUDIO_LEN);
   SecondPlay = (struct IOAudio *) GetDeviceBlock (AUDIO_LEN);
   FirstPlay->ioa_Data   = (UBYTE *)Channels;
   FirstPlay->ioa_Length = CHANNEL_SIZE;
   FirstPlay->ioa_Request.io_Message.mn_Node.ln_Pri =
SoundPrecedence;
   Open_A_Device ("audio.device",0L,&FirstPlay,01,01);
   Audio_Copy (FirstPlay,SecondPlay);
   FirstLock  = Audio_Lock (FirstPlay);
   SecondLock = Audio_Lock (SecondPlay);
   /* No sound output */
   Audio_Pervol (FirstPlay, (UWORD)0,(UWORD)0);
   for (i=0;i<WaveLength1;i++)
      FirstWave[i]   = 0;
   for (i=0;i<WaveLength2;i++)
      SecondWave[i]   = 0;
   Audio_Pervol (FirstPlay,(UWORD)1500,(UWORD)64);
   for (i=0;i<WaveLength1;i+=2) FirstWave[i] = 128;
   for (i=0;i<WaveLength2;i+=4) SecondWave[i] = 128;
   /* Calculate FirstWave  (already done ) */
   Audio_Write (FirstPlay,FirstWave,(ULONG) 320,
                                 (UWORD) 1,(BOOL) TRUE);
```

```
   while ((*bfe001 & 0x40) == 0x40)
   {
      /* Calculate SecondWave (already done) */
      Audio_Write (SecondPlay,SecondWave,(ULONG) WaveLength2,
                                        (UWORD) 1,(BOOL) TRUE);
      WaitIO (FirstPlay);
      /* FirstWave new calculation (not already done here!) */
      Audio_Write (FirstPlay,FirstWave,(ULONG) WaveLength1,
                                      (UWORD) 1,(BOOL) TRUE);
      WaitIO (SecondPlay);
   }
   Audio_Free (FirstLock);
   Audio_Free (SecondLock);
   FreeDeviceBlock (SecondPlay);
   Close_A_Device (FirstPlay);
}
/***************************************************************
*                          main()                       (User)*
***************************************************************/
main()
{
   FirstWave  = (BYTE *) AllocMem
(WaveLength1,(ULONG)MEMF_CHIP|MEMF_CLEAR);
   SecondWave = (BYTE *) AllocMem
(WaveLength2,(ULONG)MEMF_CHIP|MEMF_CLEAR);
   if (FirstWave == 0)  CloseIt ("No Memory for Wave 1 !");
   if (SecondWave == 0) CloseIt ("No Memory for Wave 2 !");
   The_Audio_Device();
   FreeMem (FirstWave,WaveLength1);
   FreeMem (SecondWave,WaveLength2);
}
```

The audio device uses commands other than those described above. These are not especially important, and you may never need them. We'll list the rest for the sake of completeness, though:

```
/***************************************************************
*                     Audio_Read()            (Audio_Support)*
* Function: Find out actual Write-Device-Block                *
*-------------------------------------------------------------*
* Input - Parameter:                                          *
* ReadRequest: Device-Block                                   *
* Channel:     Channel (0,1,2,3) whose Write-Block should     *
*              be found                                       *
*-------------------------------------------------------------*
* Return value:                                               *
* Addressof the Device-Block, for the output of the given     *
* channel or -1 if no CMD_WRITE is in operation               *
***************************************************************/
UBYTE *Audio_Read (Read_Request,Channel)
struct IOAudio    *Read_Request;
ULONG                        Channel;
{
   Read_Request->ioa_Request.io_Unit    = (struct Unit *)
Channel;
   printf ("io_Unit %ld\n",Read_Request->ioa_Request.io_Unit);
   Read_Request->ioa_Request.io_Flags   = (UBYTE)
ADIOF_SYNCCYCLE;
   Read_Request->ioa_Request.io_Command = (UWORD) CMD_READ;
   DoIO (Read_Request);
   if (Read_Request->ioa_Request.io_Error != 0)
```

```
      return ((UBYTE*)0xffffffff);
   return(Read_Request->ioa_Data);
}
```

This function gives you the address of the audio device block. The device block is necessary for sound output on a specific channel. The bit belonging to the channel is given in the channel and passed to the unit element of the audio device block. After CMD_READ, either the address of the audio device block that propels a CMD_WRITE command to the given channel or the value zero is in ioa_Data when the given channel is not currently described.

The CMD_READ function determines the sound precedence of another audio device user. This helps improve your odds of getting a channel.

ADCMD_WAITCYCLE

The above routine syncronized the execution of the command with the end of the sound output using the SYNCCYCLE flag. We can also perform this synchronization through an audio device command:

```
Do_Comand(Audio_Device_Block, (UWORD) ADCMD_WAITCYCLE);
```

This command first returns to the program when the played cycle ends.

ADCMD_RESET

This command resets the audio device to exit status:

```
Do_Command(Audio_Device_Block, (UWORD) CMD_RESET);
```

The CMD_FLUSH command is executed, the sound interrupt vectors are initialized again, and a previous CMD_STOP command is executed.

CMD_FLUSH

This command stops all current write requests and all of the listed (double buffering) write requests:

```
Do_Command(Audio_Device_Block, (UWORD) CMD_FLUSH);
```

CMD_FINISH

This command ends the sound output:

```
/*************************************************************
*                        Audio_Finish()        (Audio_Support)*
* Function: Stop sound output                                *
*------------------------------------------------------------*
* Input - Parameter:                                         *
* Audio_Device_Block: Device-Block, whose sound should be    *
*                     stopped                                *
* Sync:               Is end of sound schronized with end of *
*                     cycle                                  *
*************************************************************/
VOID Audio_Finish (Audio_Device_Block,Sync)
struct IOAudio    *Audio_Device_Block;
```

```
BOOL                                Sync;
{
   Audio_Device_Block->ioa_Request.io_Command  =
(UWORD)ADCMD_FINISH;
   if (Sync)   Audio_Device_Block->ioa_Request.io_Flags |=
(UBYTE) ADIOF_SYNCCYCLE;
   else        Audio_Device_Block->ioa_Request.io_Flags &=
(UBYTE)~ADIOF_SYNCCYCLE;

   DoIO (Audio_Device_Block);
}
```

If you want to synchronize the interruption of the actual sound with the end of the sound output, you must give the value TRUE as the `Sync` parameter in `Audio_Finish()`. Now the `SYNCCYCLE` flag is set. This flag is responsible for the synchronization. `ADCMD_FINISH` ensures that the listed write requests are acquired for execution. That is the difference between this command and `CMD_FLUSH`.

CMD_START and CMD_STOP

These commands stop (`Do_Command(Audio_Device_Block, (UWORD) CMD_STOP`) and start (`Do_Command(Audio_Device_Block, (UWORD) CMD_STOP`) sound output.

4.11.5 Sound in the interrupt code

The following audio device commands cannot be used in conjunction with interrupt level 5 or higher:

```
ADCMD_FINISH
ADCMD_FREE          (not used in the interrupt code!)
ADCMD_LOCK          (not used in the interrupt code!)
ADCMD_PERVOL
ADCMD_SETPREC       (not used in the interrupt code!)
ADCMD_WAITCYCLE
CMD_FLUSH
CMD_RESET
CMD_START
CMD_STOP
CMD_WRITE
```

This appears to be completely logical since the interrupts that announce that the DMA is done with sound output lies on level 4. The interrupts with higher CPU priorities should be reserved by the system.

4.12 The narrator device

Now we come to a derivative of the audio device: the narrator device. The narrator device makes it possible for you to access the speech synthesizer of the Amiga. Unfortunately this synthesizer only speaks one form of English. If you want to output sentences in different accents or even foreign languages, the speech comes out in plain, media English.

Through some tricks you can get the synthesizer to speak a somewhat understandable accent, or even a foreign language. The synthesizer's speech system is constructed of phoneme codes. This means that each sound has a specific phoneme code (e.g., "a", "t"). By assembling these codes in the correct order, words and sentences can be made.

Because literal translation of words into phonemes can become very complicated, Commodore supplied the translator library which contains the `translate()` routine as a single command. This routine translates all of the English words in your phoneme codes. `Translate()` can translate most of the words in the English language. Because there are also some irregularities in the English language, `translate()` also uses its own table to convert words.

`Translate()` also governs the different phonetics. Take the phrases "the car" and "the others". The word "the" in "the car" ends with a short E sound because the following word begins with a consonant. The word "the" as it appears in "the others" ends with a long E sound because the next word begins with a vowel. The phonemes are: DHAX CAA3R (the car) and DHIY AH2DHERZ (the others).

After the translator library opens (`TranslatorBase = OpenLibrary ("translator.library", 0))`, `translate()` is called with the string to be spoken, the string's length and the address of the memory region for the phoneme as well as its length:

```
char *String;                /* Ends with a null */
char Phoneme[1000];
#define LEN sizeof(Phoneme)
Error = Translate(String, strlen(String), Phoneme, LEN);
```

If enough memory exists for the phoneme, `Error` contains the value zero. If `Error` is unequal to zero, the absolute value of the error (`Error` is given with a minus sign) gives the location in the input string that can no longer be translated because of insufficient phoneme memory. The example above could have made out this location using String[-Error]. If we add this data, we can find the error:

```
Error = Translate(&String[-Error], strlen (&String[-Error]),
NewMemory, Length);
```

You can then output the phoneme string that was created from `Translate` through the narrator device. For this you must open the narrator device:

```
#define NARRAT_RB_LEN sizeof(struct narrator_rb)
struct narrator_rb *WriteRequest = 0L;
...
Open_A_Device("narrator.device", 0L, &WriteRequest, 0L,
NARRAT_RB_LEN);
```

The narrator device block (`narrator_rb`) has the following structure:

```
Offset   Structure
------   -------          struct narrator_rb
         {/* defined in "devices/narrator.h" */
0  0x00     struct IOStdReq message;          /* as always ! */
14 0x0e     UWORD           rate;          /* Words per minute */
16 0x10     UWORD            pitch;         /* basic frequency */
18 0x12     UWORD            mode;          /* human//robotics */
20 0x14     UWORD            sex;                      /* sex */
22 0x16     UBYTE       *ch_masks;    /* Channel Allocation Map */
26 0x1a     UWORD           nm_masks;      /* sizeof (ch_masks) */
28 0x1c     UWORD            volume;                /* volume */
30 0x1e     UWORD          sampfreq;     /* Sampling Frequency */
32 0x20     UBYTE            mouths;        /* Mouth form Flag */
33 0x21     UBYTE           chanmask;      /* Which channel was */
                                            /* actually used  */
34 0x22     UBYTE             numchan;      /* how many masks? */
35 0x23     UBYTE            pad;          /* for even address */
36 0x24  }
```

The narrator device uses the above device block mainly for output. For reading it uses another block:

```
Offset  Structure
------  --------          struct mouth_rb
        {/* defined in "devices/narrator.h" */
0  0x00    struct              narrator_rb voice;
36 0x24    UBYTE               width;        /* Mouth width */
37 0x25    UBYTE               height;       /* Mouth height */
38 0x26    UBYTE               shape;        /* internal ! */
39 0x27    UBYTE               pad;          /* even address */
40 0x28 }
```

As you can see from the name of this structure (`mouth`) and the comments, this structure reads the mouth form created through the spoken word.

Now we come to the narrator device commands themselves. The most important command is `CMD_WRITE`. This command allows the phoneme code to be output:

```
#define MOUTH_RB_LEN      (ULONG) sizeof(struct mouth_rb)
/***********************************************************/
/*                  Narrator_Write()      (Narrat_Support)*/
/*                                                        */
/* Function:      Narrator-Output                         */
/*--------------------------------------------------------*/
/* WriteRequest: IO-Block, through twhich the speech      */
/*               output should be directed                */
/* string:       string to be output                      */
/* rate:         Words per minute (40-400)                */
/* pitch:        Basic tone pitch (65-320)                */
/* mode:         robot (1)  or  natural (0)               */
/* sex:          Sex (0 man//1 female)                    */
/* Channels:     Channel-Map                              */
/* Size:         sizeof (Channel-Map)                     */
/* vol:          Vloume (0-64)                            */
/* freq:         Output frequency (5000-28000)            */
/* mouths:       Generate mouth form (1)                  */
/***********************************************************/

VOID Narrator_Write

(WriteRequest,string,rate,pitch,mode,sex,Channels,Size,vol,freq,
mouths)

struct narrator_rb
     *WriteRequest;
char              *string;
UWORD                     rate,pitch,mode,sex;
UBYTE                                        *Channels;
UWORD                                      Size,vol,freq,mouths;
{
   struct mouth_rb *ReadRequest;/* Request for mouth form  */
   UBYTE SpokenString[1000];    /* Phonetic memory         */
   ULONG Mouth_Rout_Count;      /* how often was           */
                                /* Mouth_Routine() called  */
   WriteRequest->rate             = rate;
   WriteRequest->pitch            = pitch;
   WriteRequest->mode             = mode;
   WriteRequest->sex              = sex;
   WriteRequest->ch_masks         = Channels;
   WriteRequest->nm_masks         = Size;
   WriteRequest->volume           = vol;
   WriteRequest->sampfreq         = freq;
   WriteRequest->mouths           = mouths;

   if (Translate (string, (ULONG) strlen
(string),SpokenString,1000L) != 0)
      CloseIt ("TranslateError");
                                  /* phonetic English string */

   WriteRequest->message.io_Data    = (APTR)  SpokenString;
   WriteRequest->message.io_Length  = (ULONG) strlen
(SpokenString);
   WriteRequest->message.io_Command = (UWORD) CMD_WRITE;

   if (mouths == 1)      /* for Mouth form generation */
```

```
   {
      ReadRequest = (struct mouth_rb *) GetDeviceBlock
(MOUTH_RB_LEN);
      Narrator_Copy (WriteRequest, ReadRequest);
                              /* Prepare ReadRequest  */
      Mouth_Init();          /* Mouth-Routine initialization */

      ReadRequest->width                     = (UBYTE) 0;
      ReadRequest->height                    = (UBYTE) 0;
      ReadRequest->voice.message.io_Command = (UWORD) CMD_READ;
      ReadRequest->voice.message.io_Error   = (UBYTE) 0;
                              /* Prepare Read command */

      if (SendIO (WriteRequest) != 0) CloseIt ("SpeakError");
                              /* send Write command */

      Mouth_Rout_Count = 0;   /* Mouth-Routine called 0 times */

      while (ReadRequest->voice.message.io_Error != ND_NoWrite)
      {
         DoIO (ReadRequest);
         Mouth_Routine (ReadRequest->width, ReadRequest-
>height,Mouth_Rout_Count);
                  /* as long as it can be read, it is read */
                  /* and Mouth_Routine() is called         */
         Mouth_Rout_Count++;
      }

      FreeDeviceBlock (ReadRequest);
      Mouth_Expunge();
                                    /* End Mouth_Routine */
   }
   else DoIO (WriteRequest);
                              /* no mouth form generation */
}                             /* only speech output       */
```

The routine joins the speech output (`CMD_WRITE`) with the discovery of the mouth form (`CMD_READ`). Next the necessary parameters are set in `WriteRequest`, and in the speech output requirement. Parameters like `Channels`, `Size` and `Vol` are self-explanatory, or were described in Section 4.11. We'll look at the other parameters used in this routine.

The first parameter to be given from `NarratorWrite()` is the narrator device block (for the writing or for the speech output) initialized from `Open_A_Device()`. The string that should be output follows in letters (not in phonemes).

`Rate` represents the number of words per minute output. The larger `rate` is, the faster the words are spoken. Values for `rate` can range from 40 to 400.

The `pitch` parameter represents the pitch at which the Amiga should speak. Values for `pitch` range from 65 to 320.

A natural voice deviates from its standard pitch (i.e., the voice's pitch raises and lowers). The `mode` parameter controls this deviation. If you want the Amiga voice to speak in a monotone fashion, set the `mode` parameter to 1. Setting the mode parameter to 0 enables natural speech.

The sex parameter specifies the gender of your computer voice. Setting sex to 1 enables the female voice, while setting sex to 0 enables the male voice.

The freq parameter controls the frequency at which speech synthesis works. The higher this frequency is, the better and more natural the output sounds. The more exact and precise the waveforms for the speech output are calculated or output, the better the result sounds. For example, with a freq of 5000 you cannot actually understand what was said. A freq equal to 28000 gives the best result (for the Amiga).

The last parameter for Narrator_Write() tells if the mouth form should be read (mouths = 1). Just as you move your mouth when speaking, you can instruct the Amiga to display a mouth on the screen as it speaks. We'll work more with the mouth at a later time. For now, let's see what happens to the speech output.

Next the variables given above (vol, freq, sex, etc.) must be added to the WriteRequest (IO block of the narrator device). Then we can translate the string to be output using Translate(). Now we need the phoneme code, ended by a null character. It was also ended with the string end character \0, in which the io_Data pointer of the WriteRequest is given as well as the length of this string in the io_Length variable of the WriteRequest.

After determining the command (WriteRequest->message.io_Command = (UWORD) CMD_WRITE;) the speech output can be started by means of DoIO(). In addition to this simple case (only speech output), Narrator_Write() offers another option for finding the mouth form. This is, as mentioned above, accessed from the Narrator_Device() by means of CMD_READ.

When you have set the parameter mouths to 1, a ReadRequest and a device block are created for reading (GetDeviceBlock()). Then the variables io_Unit and io_Device are copied from the original device block into the ReadRequest. Here are the first two lines of the function's definition:

```
VOID Narrator_Copy (Old_Request, New_Request)
struct narrator_rb *Old_Request;
struct mouth_rb              *New_Request;
{...}
```

After the ReadRequest the routine Mouth_init() is called. This routine belongs to the Mouth_Init(), Mouth_Routine() and Mouth_Expunge() commands. These routines are required for mouth formation. For example, in `Mouth_Init()` screens and windows are opened in which the mouth movement from the `Mouth_Routine()` is drawn. When the speech output ends, the screens and windows opened from `Mouth_Init()` are closed again from `Mouth_Expunge()`.

Before we call `Mouth_Routine()` or `Mouth_Expunge()`, we must first process the `ReadRequest`. Next the mouth form must be initialized. The mouth form given by the width and height of the mouth must be set to zero. Next we assign the `ReadRequest` block its task, speaking the command to be executed (`CMD_READ`).

Now we read the mouth form until the previous `Write` command ends, and also until the last word is spoken. When the narrator device has nothing more to say, the error `ND_NoWrite` is encountered when trying to read more mouth forms. `CMD_READ` is a synchronized command and should be called through `DoIO()`. `DoIO()` returns with `CMD_READ` if a mouth form change is encountered.

When such a change has been found, the routine `Mouth_Routine()` is called in `Narrator_Write()`, and the width, height and the number of the call of this routine is given. The following program uses the previously prepared narrator write routine:

```
/***********************************************/
/*            The-Narrator-Device              */
/*                                             */
/*           (c) Bruno Jennrich                */
/***********************************************/
/***********************************************/
/* Compile   info:                             */
/*---------------------------------------------*/
/* cc Say.c                                    */
/* ln Narrat_Support.o Say.o Devs_Support.o -lc */
/***********************************************/
#include "exec/types.h"
#include "exec/types.h"
#include "exec/memory.h"
#include "exec/devices.h"
#include "intuition/intuition.h"
#include "intuition/intuitionbase.h"
#include "graphics/gfxbase.h"
#include "graphics/gfxmacros.h"
#include "libraries/translator.h"
#include "Narrat_Support.h"
#include "devices/narrator.h"
#define WIDTH  150        /* size of the mouth window */
#define HEIGHT 75
VOID *OpenLibrary();                 /* functions used */
VOID *OpenWindow();
struct Library       *TranslatorBase = 0L; /* Libraries*/
struct IntuitionBase *IntuitionBase  = 0L;
struct GfxBase       *GfxBase        = 0L;
struct NewWindow      NewWindow;        /* Window Defs */
struct Window        *Window         = 0L;
```

```
struct narrator_rb   *WriteRequest   = 0L; /* Narrator device
                                              blocks */
UBYTE Channels[4] = {3, 5, 10, 12};      /* Channel allocations
                                            mask */
                                         /* see Audio Device   */
#define MOUTH_RB_LEN     (ULONG) sizeof(struct mouth_rb)
ULONG Mouth_Rout_Count;                /* how often was */
                             /* Mouth_Routine() called */
/*********************************************************/
/*                       CloseIt()               (User)*/
/*                                                     */
/* Function:  Free all of the allocated structures     */
/*-----------------------------------------------------*/
/* String:    Error message                            */
/*********************************************************/
VOID CloseIt(String)
char        *String;
{
   UWORD Error = 0, i;
   UWORD *dff180 = (UWORD *)0xdff180; /* background color
                                         register */
   if (strlen(String) > 0)
   {
      for (i=0;i<0xffff;i++)  *dff180 = i;/* screen blink */
      puts(String);                 /* display string */
      Error = 100;          /* Error Code != 0 (EXIT()) */
   }
   if ((WriteRequest != 0L) && (WriteRequest->message.io_Device
!= 0L))
      Close_A_Device(WriteRequest);
   if (TranslatorBase != 0L)
      CloseLibrary(TranslatorBase);
   exit(Error);
}
/*********************************************************/
/*                    Mouth_Init()               (User)*/
/* Function:  Mouth initialization routine             */
/*********************************************************/
VOID Mouth_Init()
{
   if ((GfxBase = (struct GfxBase *)
OpenLibrary("graphics.library", 0L)) == 0)
      CloseIt("No Graphics !!!");
   if ((IntuitionBase = (struct IntuitionBase *)
        OpenLibrary("intuition.library", 0L)) ==0)
      CloseIt("No Intuition !!!");
   NewWindow.LeftEdge     = 640/2-WIDTH/2;
   NewWindow.TopEdge      = 200/2-HEIGHT/2;
   NewWindow.Width        = WIDTH;
   NewWindow.Height       = HEIGHT;
   NewWindow.DetailPen    = 0;
   NewWindow.BlockPen     = 1;
   NewWindow.IDCMPFlags   = 0;
   NewWindow.Flags        = ACTIVATE | RMBTRAP | NOCAREREFRESH;
   NewWindow.FirstGadget  = (struct Gadget *) 0L;
   NewWindow.CheckMark    = (struct Image *)  0L;
   NewWindow.Title        = (UBYTE *) " Mouth-Shape";
   NewWindow.Screen       = (struct Screen *) 0L;
   NewWindow.BitMap       = (struct BitMap *) 0L;
   NewWindow.MinWidth     = 0;
   NewWindow.MaxWidth     = 0;
   NewWindow.MinHeight    = 0;
```

```
   NewWindow.MaxHeight    = 0;
   NewWindow.Type         = WBENCHSCREEN;
   if ((Window = (struct Window *) OpenWindow(&NewWindow)) == 0)
      CloseIt("No Window !!!");
   SetAPen(Window->RPort, 2);
   SetDrMd(Window->RPort, (ULONG)(COMPLEMENT|JAM1));
}
/*********************************************************/
/*                    Mouth_Expunge()            (User)*/
/*                                                      */
/* Function:  Mouth Expunge Routine                     */
/*********************************************************/
VOID Mouth_Expunge()
{
   if (Window != 0L)        CloseWindow(Window);
   if (GfxBase != 0L)       CloseLibrary(GfxBase);
   if (IntuitionBase != 0L) CloseLibrary(IntuitionBase);
}
/*********************************************************/
/*                    Mouth_Routine()            (User)*/
/*                                                      */
/* Function:  Mouth Routine                             */
/*------------------------------------------------------*/
/* width:     Mouth form width                          */
/* height:    Mouth form height                         */
/*********************************************************/
VOID Mouth_Routine(width, height, Count)
UBYTE              width, height;
ULONG                             Count;
{
   static ULONG old_width,                 old_height;
   if (Count != 0L)
   {
      /* erase old shape */
      Move(Window->RPort, (ULONG)(WIDTH/2),   (ULONG)(HEIGHT/2-
height));
      Draw(Window->RPort, (ULONG)(WIDTH/2),
(ULONG)(HEIGHT/2+height));
      Move(Window->RPort, (ULONG)(WIDTH/2-width),
(ULONG)(HEIGHT/2));
      Draw(Window->RPort, (ULONG)(WIDTH/2+width),
(ULONG)(HEIGHT/2));
   }
   /* draw new shape*/
   width *= 2;
   WaitTOF();
   Move(Window->RPort, (ULONG)(WIDTH/2),   (ULONG)(HEIGHT/2-
height));
   Draw(Window->RPort, (ULONG)(WIDTH/2),
(ULONG)(HEIGHT/2+height));
   Move(Window->RPort, (ULONG)(WIDTH/2-width),
(ULONG)(HEIGHT/2));
   Draw(Window->RPort, (ULONG)(WIDTH/2+width),
(ULONG)(HEIGHT/2));
   old_width  = (ULONG)width;
   old_height = (ULONG)height;
}
/*****************************************************/
/*             The_Narrator_Device()        (User)*/
/*                                                  */
/* Function: use narrator device                    */
/*--------------------------------------------------*/
```

```
/* Input Parameters:                                */
/*                                                  */
/* string:   output string                          */
/* rate:     Words per minute                       */
/* pitch:    basic frequency                        */
/* mode:     robotic or human                       */
/* sex:      sex                                    */
/* vol:      volume                                 */
/* freq:     Sampling frequency                     */
/* mouths:   generate mouth form?                   */
/****************************************************/
VOID The_Narrator_Device(string, rate, pitch, mode, sex, vol,
freq, mouths)
char                   *string;
UWORD                 rate, pitch, mode, sex, vol, freq, mouths;
{
   TranslatorBase = OpenLibrary("translator.library", 0L);
   if (TranslatorBase == 0L) CloseIt("NoTranslatorBase");
   Open_A_Device("narrator.device", 0L, &WriteRequest, 0L,
MOUTH_RB_LEN);
   Narrator_Write
    (WriteRequest, string, rate, pitch, mode, sex, Channels,
(UWORD)sizeof (Channels), vol, freq, mouths);
   Close_A_Device(WriteRequest);
   CloseLibrary(TranslatorBase);
}
/****************************************************/
/*                 AtoUWORD()                (User)*/
/*                                                  */
/* Function: change number string (ASCII) after UWORD*/
/*--------------------------------------------------*/
/* Input Parameters:                                */
/*                                                  */
/* Bufpointer: number string to change              */
/*--------------------------------------------------*/
/* Return value: value of the numeral string        */
/****************************************************/
UWORD AtoUWORD(BufPointer)
char           *BufPointer;
{
   UWORD Result;
   Result = 0;
   while (*BufPointer == ' ') BufPointer++;  /* read over Spaces
*/
   while (*BufPointer != '\000')
   {
      Result *= 10;
      Result += *(BufPointer++) - '0';
   }
   return(Result);
}
/****************************************************/
/*                    main()                        */
/****************************************************/
main (argc, argv)
UWORD argc;
char     **argv;
{
   UWORD Param[7];
   ULONG i;
   Param[0] = DEFRATE;     /* rate      */ /* Defaults */
   Param[1] = DEFPITCH;    /* pitch     */
```

```
   Param[2] = DEFMODE;      /* mode      */
   Param[3] = DEFSEX;       /* sex       */
   Param[4] = DEFVOL;       /* volume    */
   Param[5] = DEFFREQ;      /* frequency */
   Param[6] = 1;            /* mouths    */
   if ((argc < 2) || (argc > 9))
      printf("Usage: SAY \"string\" [rate] [pitch] [mode] [sex]
[volume] [sampfreq] [mouths]\n");
   else
   {
      for (i=0;i<(argc-2);i++)
         Param[i] = AtoUWORD (argv[i+2]);
      The_Narrator_Device
         (argv[1], Param[0], Param[1], Param[2], Param[3],
Param[4], Param[5], Param[6]);
   }
}
```

The following routine lists still more narrator support routines. From these you can see which commands the narrator device supports:

```
/************************************************************/
/*                  Narrator_Copy()       (Narrat_Support)*/
/*                                                          */
/* Function:  Narrator-Device-Block copy                   */
/*---------------------------------------------------------*/
/* Old_Request: Original IO-Block                          */
/* New_Request: Copy of the IO-Blocks                      */
/************************************************************/

VOID Narrator_Copy (Old_Request, New_Request)
struct narrator_rb *Old_Request;
struct mouth_rb                 *New_Request;
{
   New_Request->voice.message.io_Device =    /* you only need to
copy    */
     Old_Request->message.io_Device;         /* 4io_Device4and
4io_Unit4 */
   New_Request->voice.message.io_Unit =
     Old_Request->message.io_Unit;
}


/************************************************************/
/*                  Narrator_Stop()       (Narrat_Support)*/
/*                                                          */
/* Function:     Stop Narrator output                      */
/*---------------------------------------------------------*/
/* WriteRequest: IO-Block, whose output should be stopped */
/************************************************************/

VOID Narrator_Stop (WriteRequest)
struct narrator_rb *WriteRequest;
{
   WriteRequest->message.io_Command = (UWORD) CMD_STOP;
   DoIO (WriteRequest);
}


/************************************************************/
/*                  Narrator_Start()      (Narrat_Support)*/
/*                                                          */
```

```
/* Function:     Start Narrator output                     */
/*---------------------------------------------------------*/
/* WriteRequest: IO-Block, whose output should be started */
/*               (after Narrator_Stop())                   */
/***********************************************************/

VOID Narrator_Start (WriteRequest)
struct narrator_rb *WriteRequest;
{
   WriteRequest->message.io_Command = (UWORD) CMD_START;
   DoIO (WriteRequest);
}


/***********************************************************/
/*                   Narrator_Flush()      (Narrat_Support)*/
/*                                                         */
/* Function:     End Narrator output                       */
/*---------------------------------------------------------*/
/* WriteRequest: IO-Block, whose outpur should be ended   */
/***********************************************************/

VOID Narrator_Flush (WriteRequest)
struct narrator_rb *WriteRequest;
{
   WriteRequest->message.io_Command = (UWORD) CMD_FLUSH;
   DoIO (WriteRequest);
}


/***********************************************************/
/*                   Narrator_Reset()      (Narrat_Support)*/
/*                                                         */
/* Function:     Narrator-Device rest to known condition  */
/*---------------------------------------------------------*/
/* WriteRequest: IO-Block, through which the Narrator     */
/*               Device is reset                           */
/***********************************************************/

VOID Narrator_Reset (WriteRequest)
struct narrator_rb *WriteRequest;
{
   WriteRequest->message.io_Command = (UWORD) CMD_RESET;
   DoIO (WriteRequest);
}
```

The Narrator_Support.h file is listed below:

```
VOID Narrator_Copy();
VOID Narrator_Stop();
VOID Narrator_Start();
VOID Narrator_Flush();
VOID Narrator_Reset();
VOID Narrator_Write();
```

4.13 The timer device

The timer device accesses the Amiga's internal timer. This timer serves two purposes. You can measure time through the vertical blank, or you can measure time through the CIA timer. Both have advantages and disadvantages: The vertical blank timer is constant over a long period of time. It has accuracy of only a 50th of a second.

The CIA timer has an accuracy of a microsecond. The disadvantage is that it is inaccurate over a long period of time. The user must decide for himself which timer to use. The timer to be used is established by the `OpenDevice()` command. The `unit` parameter specifies which timer you want to use:

```
UNIT_MICROHZ 0 (CIA Timer)
UNIT_VBLANK  1 (Vertical Blank Timer)
```

The vertical blank timer is opened as follows:

```
struct timerequest *TimeRequest = 0L;
#define TIME_LEN (ULONG) (sizeof (struct timerequest))
...
   Open_A_Device("timer.device", (ULONG) UNIT_VBLANK,
                 &TimeRequest, TIME_LEN);
...
```

The timer device supports the following primary commands:

`TR_ADDREQUEST`	wait for a determined time
`TR_GETSYSTIME`	get system time
`TR_SETSYSTIME`	set system time

The timer device also supports three additional commands:

`AddTime()`	0x2a
`SubTime()`	-0x30
`CmpTime()`	-0x36

The following routine allows a predetermined time to elapse:

```
/****************************************************************
*                       WaitTime()            (Timer_Support)*
* Function: Wait for a predetermined amount of time             *
*--------------------------------------------------------------*
* Input - Parameter:                                            *
* TimeRequest: Timer-Device-Block                               *
* Secs,Micro:  Wait for how many seconds and microseconds?      *
****************************************************************/
```

```
VOID WaitTime (TimeRequest,Secs,Micro)
struct timerequest
              *TimeRequest;
ULONG                       Secs,Micro;
{
   TimeRequest->tr_time.tv_secs    = Secs;
   TimeRequest->tr_time.tv_micro   = Micro;

   TimeRequest->tr_node.io_Command = TR_ADDREQUEST; /* Command
*/
   DoIO (TimeRequest);
}
```

The seconds and microseconds are given to the timer device block as parameters, and the `TR_ADDREQUEST` command is called. This routine first returns when the given time elapses. Remember that with `UBIT_VBLANK` the exact statement of microseconds is meaningless. Enter the value zero here.

Let's take a closer look at the timer request structure:

```
Offset       Structure
------       --------             struct timerequest
             {
0  0x00         struct IORequest tr_node;
32 0x20         struct timeval   tr_time;
40 0x28      } /* defined in "devices/timer.h" */
```

The `timeval` structure contained in this structure looks like the following:

```
Offset       Structure
------       --------             struct timeval
             {
0  0x00         ULONG tv_secs;
4  0x04         ULONG tv_micro;
8  0x08      }  /* defined in "devices/timer.h" */
```

This structure contains all of the times to be set or read. For example, if you want to read the system time, it is transferred to these variables:

```
/*************************************************************
*                           GetSysTime()        (Timer_Support)*
* Function: Find out system time                              *
*-------------------------------------------------------------*
* Input - Parameter:                                          *
* TimeRequest: Timer-Device-Block                             *
*-------------------------------------------------------------*
* Return value:                                               *
* Pointer to Timeval structure of the Timer-Device-Block      *
*************************************************************/

struct timeval *GetSysTime (TimeRequest)
struct timerequest          *TimeRequest;
{
   Do_Command (TimeRequest,TR_GETSYSTIME);
   return (&TimeRequest->tr_time);
}
```

You get the `timeval` structure of the device block with the actual system time with `Time1 = (struct timeval *) GetSysTime (timeRequest)`. The system time returns the current system time and the current date.

There's no problem calculating the system time with the realtime clock. What does the user do if he doesn't have such a clock? He must always set the time and date by hand. When writing to disk, the time of this disk change is saved to the boot block, which is read with every boot and declares the current system time. Naturally many inconsistencies are encountered which you should be able to fix with the help of CLI commands.

Now we come to how the system is decoded. The internal clock counts in seconds, starting from January 1, 1978. When you want to display the system (given in seconds and microseconds) in a format that the user can understand, some calculations must be made:

```
ULONG Days_of_Months[] = {31l, /* Jan */ /* How many days does*/
                          28l, /* Feb */ /* each month have ? */
                          31l, /* Mdr */  /* (Timer Support)  */
                          30l, /* Apr */
                          31l, /* Mai */
                          30l, /* Jun */
                          31l, /* Jul */
                          31l, /* Aug */
                          30l, /* Sep */
                          31l, /* Okt */
                          30l, /* Nov */
                          31l  /* Dez */};
char *Days_of_Week[] = {"Sunday   ",        /* Week day names */
                        "Monday   ",
                        "Tuesday  ",
                        "Wednesday",
                        "Thursday ",
                        "Friday   ",
                        "Saturday "};
/****************************************************************
*                        LeapYear()              (TimerSupport)*
* Function: Is year a LeapYear?                                 *
*---------------------------------------------------------------*
* Input - Parameter:                                            *
* Year: Year to select                                          *
*---------------------------------------------------------------*
* Return value:                                                 *
* TRUE:  Year IS LeapYear                                       *
* FALSE: Year is NOT LeapYear                                   *
****************************************************************/
BOOL LeapYear (Year)
ULONG Year;
{
   if (((Year/4l)*4l) == Year) &&
      (((Year/100l)*100l) != Year)) return (TRUE);
   else return (FALSE);
      /* When year is divisible by  4, but nor by */
      /* 100, it is a  Leap Year */
}
```

```
/*************************************************************
*                    MakeMonthTable()         (TimerSupport)*
* Function: Update month day table (29 or 28. February?)    *
*-----------------------------------------------------------*
* Input - Parameter:                                        *
* Year:  Year to select                                     *
* Table: Address of th Month day table (i.e. Days_of_Month)  *
*************************************************************/
VOID MakeMonthTable (Year,Table)
ULONG                Year;
ULONG                    *Table;
{
   if (LeapYear (Year))
      Table[1] = 29;
   else
      Table[1] = 28;
   /* If year u sleapyear, set February to  29 days */
   /* instead of 28                                 */
}
/*************************************************************
*                     SysTime_to_TimeDate()     (TimerSupport)*
* Function: Transfer system time to Timedate structure      *
*-----------------------------------------------------------*
* Input - Parameter:                                        *
* TimeRequest: Timer-Device-Block                           *
* TimeDate:    TimeDate structure to fill                   *
*************************************************************/
VOID SysTime_to_TimeDate (TimeRequest,TimeDate)
struct timerequest       *TimeRequest;
struct TimeDate                      *TimeDate;
{
   struct timeval *SysTime;       /* for GetSysTime */
   ULONG  SysTimeSecs;
   ULONG  all_Days;               /* Days since 1.1 1978 */
   ULONG  year_Days;              /* Days in year        */
   UWORD  leap_year, years;       /* Loop variables      */
   ULONG  month;
   SysTime = GetSysTime(TimeRequest);      /* Get system time */
   SysTimeSecs = SysTime->tv_secs;         /* get seconds  */
   if (SysTimeSecs>Secs_1980)             /* later than 1980 ? */
   {
      SysTimeSecs -= Secs_1980;   /* Yes,  then subtract       */
      TimeDate->Year = 19801;     /* (1980-1978=2) year from
system */
      all_Days = (ULONG)(2*365);  /* Set year to 1980 and the */
                                  /* number of days past since*/
   }                        /* "zero hours" (1/1/1978) at 2*365 */
   else
   {
      all_Days = 01;              /* Time smaller than 1980   */
      TimeDate->Year = 19781; /* year = 1978, days elasped = 0*/
      if (SysTimeSecs > SecondsPerYear)
      {                                       /* 1979 ? */
         SysTimeSecs-= SecondsPerYear;    /* yes */
         all_Days += 3651;
         TimeDate->Year++;
      }
      goto Get_Month;                    /* since year i sknown */
   }                                     /* calculate new month */
   for (leap_year = 0; leap_year < 34; leap_year++)
   {                                    /* calculate exact year*/
     if (SysTimeSecs >= SecondsPerLeapYear)/*(later than 1980)*/
```

```
      {
         SysTimeSecs   -= SecondsPerLeapYear;  /* a LeapYear */
         TimeDate->Year++;
         all_Days += 366l;
      }
      for (years = 0; years < 3; years++)
      {                                     /* three normal years */
         if (SysTimeSecs >= SecondsPerYear)
         {
            SysTimeSecs -= SecondsPerYear;
            TimeDate->Year++;
            all_Days += 365l;
         }
      }
   }
   MakeMonthTable (TimeDate->Year,Days_of_Months);
                                  /* February = 28 or 29 days?  */
                                  /* ! 1978 & 1979 no LeapYear!*/
Get_Month:
   year_Days = 0l;                    /* day of the year = 0 */
   for (month = 0l; month < 11l; month++)
   {
      if (SysTimeSecs >= (Days_of_Months[month]*SecondsPerDay))
      {                                     /* calculate month */
         SysTimeSecs   -= (Days_of_Months[month]*SecondsPerDay);
         all_Days      +=  Days_of_Months[month];
         year_Days     +=  Days_of_Months[month];
      }
      else break;
   }
   TimeDate->Month = month+1; /* Otherwiset 0 = January etc. */
   TimeDate->Day = (SysTimeSecs/SecondsPerDay)+1l;
/* day 0 = first.        Month is first */
   SysTimeSecs   -= (SysTimeSecs/SecondsPerDay)*SecondsPerDay;
   year_Days     += TimeDate->Day-1l;      /* exp:  02.02. all*/
   all_Days      += TimeDate->Day-1l;      /* 32 days elapsed */
   TimeDate->Hour =  SysTimeSecs/SecondsPerHour;       /* hour */
   SysTimeSecs    -= (SysTimeSecs/SecondsPerHour)*SecondsPerHour;
   TimeDate->Mins =  SysTimeSecs/SecondsPerMinute;  /* Minute */
   SysTimeSecs    -=
(SysTimeSecs/SecondsPerMinute)*SecondsPerMinute;
   TimeDate->Secs = SysTimeSecs;                  /* Second */
   TimeDate->Week = year_Days/7;           /* week of the year*/
   TimeDate->Week_Day = all_Days % 7;
}                    /* Weekday (1/1/1978 waa a Sunday (0)) */
```

Of these three routines only the `System_to_TimeDate()` function is of importance. This routine fills a structure given by us, which can be easily interpreted:

```
Offset    Structure
------    --------         struct TimeDate
          {
0  0x00     ULONG Year;         /* year */
4  0x04     ULONG Month;        /* Month */
8  0x08     ULONG Day;          /* day of the month */
12 0x0c     ULONG Week;         /* week of the year */
16 0x10     ULONG Week_Day;     /* Week day (0==Sonntag etc.) */
20 0x14     ULONG Hour;         /* hour */
24 0x18     ULONG Mins;         /* Minute */
28 0x2c     ULONG Secs;         /* Seconds */
```

Program description

After the system time loads and the program is instructed to read seconds instead of microseconds, the routine checks for a current system date earlier than 1980. Because 1980 is a leap year, a loop searches through proceeding years.

To find out the number of years that have elapsed, the program subtracts the number of seconds that have elapsed in a year from the current system time until the result of this subtraction is less than a year. Leap years are determined using the `Leapyear()` function. Once you determine the correct year, you can calculate the month. Because the 12 months in a year have different number of days, the subtraction from the current system time becomes more complicated. If a year is a leap year, the `MakeMonth()` routine sets the number of days in February to 29 rather than 28.

Dividing the remaining number of seconds by the number of seconds in a day gives us the day of the month. This method is adapted to find the hours, minutes and seconds. When subtracting seconds you subtract the calculated number of days/hours/minutes/etc. The remaining seconds give the seconds within the current minute.

Adding the number of elapsed days gives us the weekday. Next the program divides the number of days elapsed since January 1, 1978 by seven; the remainder gives the current weekday. You can display the corresponding weekday with `printf("%s\n", Days_of_Week [TimeDate->WeekDay]);`.

We needed the following constants for the calculations in the above routine:

```
#define SecondsPerMinute    (ULONG)(60L)
#define SecondsPerHour      (ULONG)(60L*60L)
#define SecondsPerSay       (ULONG)(60L*60L*24L)
#define SecondsPerYear      (ULONG)(60L*60L*24L*365L)
#define SecondsPerLeapYear  (ULONG)(60L*60L*24L*366L) /* leap
                                                          year */
#define Secs_1980           (ULONG)(2L*SecondsPerYear)
                                    /* Seconds from */
                                    /* 1.1 1978 up */
                                    /* to 1.1 1980 */
```

We also wrote a routine to perform the opposite function. The new system time is calculated and set from a given `Timedate` structure:

```
/****************************************************************
*                     TimeDate_to_SysTime()    (TimerSupport)*
* Function: TimeDate becomes system time                        *
*---------------------------------------------------------------*
* Input - Parameter:                                            *
* TimeRequest: Timer-Device-Block                               *
```

```
* TimeDate:    TimeDate-Structure, to become system time       *
****************************************************************/

BYTE TimeDate_to_SysTime (TimeRequest,TimeDate)
struct timerequest       *TimeRequest;
struct TimeDate                      *TimeDate;
{
   ULONG i;
   ULONG SysTimeSecs=0l;
   if (TimeDate->Hour > 24) return (TDERR_HOUR_OUT_OF_RANGE);
   if (TimeDate->Mins > 59) return (TDERR_MINS_OUT_OF_RANGE);
   if (TimeDate->Secs > 59) return (TDERR_SECS_OUT_OF_RANGE);
   if ((TimeDate->Year < 1978l) && (TimeDate->Year > 2114l))
      return (TDERR_YEAR_OUT_OF_RANGE);
                                          /* TimeDate test */
   if (TimeDate->Month > 12)
      return (TDERR_MONTH_OUT_OF_RANGE);
   if ((TimeDate->Day > Days_of_Months[TimeDate->Month-1]) ||
       (TimeDate->Day == 0))
       return (TDERR_DAY_OUT_OF_RANGE);
   SysTimeSecs = TimeDate->Hour*SecondsPerHour+
                 TimeDate->Mins*SecondsPerMinute+
                 TimeDate->Secs+
                 (TimeDate->Day-1)*SecondsPerDay;
                             /* hours, minutes, seconds and */
                             /* days change after seconds   */
   MakeMonthTable (TimeDate->Year,Days_of_Months);
                                     /* February 28 or 29 days? */
   for (i=0l;i<(TimeDate->Month-1l);i++)
      SysTimeSecs += Days_of_Months[i]*SecondsPerDay;
                                         /* Month after seconds */
   for (i=1978l;i<TimeDate->Year;i++)
   if (LeapYear (i)) SysTimeSecs += SecondsPerLeapYear;
   else              SysTimeSecs += SecondsPerYear;
                                          /* Year after seconds */
   SetSysTime(TimeRequest,SysTimeSecs,0l);
}                                      /* Set new system time */
```

This routine multiplies the values of the `Timedate` structure by the number of seconds for the year, day, month, etc. and adds these products, performing any compensation needed for leap years. The calculated system time is reset:

```
/****************************************************************
*                        SetSysTime()            (TimerSupport)*
* Funktion: Set system time                                     *
*--------------------------------------------------------------*
* Input - Parameter:                                            *
* TimeRequest: Timer-Device-Block                               *
* Secs,Micro: New system time in seconds and microseconds       *
****************************************************************/
VOID SetSysTime (TimeRequest,Secs,Micro)
struct timerequest
               *TimeRequest;
ULONG                       Secs,Micro;
{
   TimeRequest->tr_time.tv_secs  = Secs;
   TimeRequest->tr_time.tv_micro = Micro;
   TimeRequest->tr_node.io_Command = TR_SETSYSTIME;
   DoIO  (TimeRequest);
```

```
}
```

You've now seen the most common timer device commands. Let's look at the remaining three device commands (`SubTime()`, `AddTime()` and `CmpTime()`). These commands always supply two `timeval` structures which contain two times (for example, `SubTime (timeval1,timeval2);`). The names of these commands contain these results. `SubTime()` subtracts the time of the second `timeval` structure from the first, and saves the result in the first `timeval` structure.

`AddTime()` adds the two `timeval` structures and places the result in the first `timeval` structure. `CmpTime()` compares the two given `timeval` structures. The result of this function is zero for equality, >0 when `timeval1` > `timeval2`, and <0 when `timeval1` < `timeval2`. You must first open `TimerBase` before this function can be called:

```
struct Device *TimerBase;
...
   TimerBase = TimerRequest->tr_node.io_Device;
```

Now you know the basics of the `AddTime()`, `SubTime()`, and `CmpTime()` routines. The following program applies these to the system time:

```
/****************************************************************
*                         Timer-Device                          *
*                      (c) Bruno Jennrich                       *
*                          Juni 1988                            *
*  Compile-Info: (TimerComp)                                    *
*  cc Timer                                                     *
*  ln Timer.o Timer_Support.o Devs_Support.o -lc                *
****************************************************************/
#include "exec/types.h"
#include "exec/nodes.h"
#include "exec/lists.h"
#include "exec/memory.h"
#include "exec/ports.h"
#include "exec/libraries.h"
#include "exec/io.h"
#include "exec/devices.h"
#include "devices/timer.h"
#include "Timer_Support.h"
VOID *GetSysTime();
extern ULONG  Days_of_Month[];
extern char  *Days_of_Week[];
struct TimeDate TimeDate;
struct timerequest *TimeRequest=0l;
struct timeval     *SystemTime=0l,
                    OneDay = {(ULONG)SecondsPerDay,0l};
struct Device *TimerBase;
/****************************************************************
*                          CloseIt()                     (User)*
* Function: Display erro rand close everything                  *
*--------------------------------------------------------------*
```

```
* Input - Parameter:                                           *
* String: Error-String                                         *
****************************************************************/
VOID CloseIt (String)
char           *String;
{
   UWORD i;
   UWORD *dff180 = (UWORD *)0xdff180;
   UWORD Error   = 0;
   if (strlen (String) > 0)
   {
      for (i=0;i<0xffff;i++)  *dff180 = i;
      puts (String);
      Error = 100;
   }
   if (TimeRequest != 0l)     Close_A_Device (TimeRequest);
   exit (Error);
}
/****************************************************************
*                        TimePrintout()                  (User)*
* Function: Display TimeDate-Structure (Time)                  *
*--------------------------------------------------------------*
* Input - Parameter:                                           *
* TimeDate: TimeDate structure to dislpay                      *
****************************************************************/
VOID TimePrintout (TimeDate)
struct TimeDate   *TimeDate;
{
   printf ("%s  %02ld %02ld %04ld  Time: %02ld:%02ld:%02ld\n",
            Days_of_Week[TimeDate->Week_Day],
            TimeDate->Month,
            TimeDate->Day,
            TimeDate->Year,
            TimeDate->Hour,
            TimeDate->Mins,
            TimeDate->Secs);
}
/****************************************************************
*                        The_Timer_Device()              (User)*
* Function: Timer-Device and Timer-Support support             *
****************************************************************/
VOID The_Timer_Device()
{
   Open_A_Device ("timer.device",(ULONG) UNIT_VBLANK,
                  &TimeRequest,0l,TIME_LEN);
   TimerBase = TimeRequest->tr_node.io_Device;
   printf ("System Time is:\n");
   SysTime_to_TimeDate (TimeRequest,&TimeDate);
   TimePrintout (&TimeDate);
   printf ("Now add one Day to System Time\n");
   SystemTime = (struct timeval*) GetSysTime(TimeRequest);
   AddTime (SystemTime,&OneDay);
   SetSysTime (TimeRequest,SystemTime->tv_secs,SystemTime-
>tv_micro);
   printf ("System Time now is:\n");
   SysTime_to_TimeDate (TimeRequest,&TimeDate);
   TimePrintout (&TimeDate);
   printf ("Subtract the Day from System Time and wait 15
Seconds\n");
   SubTime (SystemTime,&OneDay);
   SetSysTime (TimeRequest,SystemTime->tv_secs,SystemTime-
>tv_micro);
```

```
   WaitTime (TimeRequest,15l,0l);
   printf ("This is Northern-System-Time:\n");
   SysTime_to_TimeDate (TimeRequest,&TimeDate);
   TimePrintout (&TimeDate);
   Close_A_Device (TimeRequest);
}
/****************************************************************
*                          main()                               *
****************************************************************/
main()
{
   The_Timer_Device();
}
```

The following is the Timer_Support.h file used by the Timer.c program:

```
/****************************************************************
*                      Timer_Support.h                          *
* Include File for Timer_Support.c and the timer program        *
****************************************************************/
#define SecondsPerMinute     (60l)
#define SecondsPerHour       (60l*60l)
#define SecondsPerDay        (60l*60l*24l)
#define SecondsPerYear       (60l*60l*24l*365l)    /* Year */
#define SecondsPerLeapYear   (60l*60l*24l*366l)    /* Leapyear */
#define Secs_1980        (2l*SecondsPerYear)  /* Seconds from */
                                              /* 1.1 1978 to */
                                             /*  1.1 1980 */
#define TDERR_HOUR_OUT_OF_RANGE  -2 /* Timer-Device Errors */
#define TDERR_MINS_OUT_OF_RANGE  -3
#define TDERR_SECS_OUT_OF_RANGE  -4
#define TDERR_YEAR_OUT_OF_RANGE  -5
#define TDERR_MONTH_OUT_OF_RANGE -6
#define TDERR_DAY_OUT_OF_RANGE   -7
#define TIME_LEN (ULONG) sizeof (struct timerequest)
                        /* Size of Device-Block */
struct TimeDate {                   /* Structure */
                ULONG Year;         /* Yearr */
                ULONG Month;        /* Monht */
                ULONG Day;          /* Month day */
                ULONG Week;         /* Week of year */
                ULONG Week_Day;     /* Weekday (0=Sunday etc) */
                ULONG Hour;         /* hours */
                ULONG Mins;         /* minutes */
                ULONG Secs;         /* seconds */
               };
```

4.14 The trackdisk device

The trackdisk device controls the Amiga's disk drives. This device uses another `IOStdReq` block for its operation (we'll discuss the variables used in this structure later):

```
Offset   Structure
-------  --------         struct IOExtTD
         {
0  0x00     struct IOStdReq iotd_Req;
48 0x30     ULONG           iotd_Count;
52 0x34     ULONG           iotd_SecLabel;
56 0x38  } /* defined in "devices/trackdisk.h" */
```

The trackdisk device supports the following commands:

CMD_READ	(2)	read sector
CMD_WRITE	(3)	write sector in TrackBuffer
CMD_UPDATE	(4)	write TrackBuffer to disk
CMD_CLEAR	(5)	declare TrackBuffer invalid
TD_MOTOR	(9)	turn motor on/off
TD_SEEK	(10)	position read/write head
TD_FORMAT	(11)	format track
TD_REMOVE	(12)	install media change interrupt
TD_CHANGENUM	(13)	determine disk change number
TD_CHANGESTATE	(14)	check for inserted disk
TD_PROTSTATUS	(15)	test write protect
TD_RAWREAD	(16)	read track (raw data)
TD_RAWWRITE	(17)	write track (raw data)
TD_GETDRIVETYPE	(18)	determine drive type (3-1/2" or 5-1/4")
TD_GETNUMTRACKS	(19)	determine number of tracks
TD_ADDCHANGEINT	(20)	add media change interrupt
TD_REMCHANGEINT	(21)	remove media change interrupt
TD_LASTCOMM	(22)	determine last command executed (not currently implemented)

The trackdisk device includes extended commands, which set command number bit 15. These commands perform the same functions as the ones listed above, except that these commands don't execute immediately after a disk change:

ETD_READ	(32770)	read sector
ETD_WRITE	(32771)	write sector in TrackBuffer
ETD_UPDATE	(32772)	write TrackBuffer to disk
ETD_CLEAR	(32773)	TrackBuffer declared invalid

ETD_MOTOR	(32777)	turn motor on/off
ETD_SEEK	(32778)	position read/write head
ETD_FORMAT	(32779)	format track
ETD_RAWREAD	(32784)	read track (raw data)
ETD_RAWWRITE	(32785)	write track (raw data) (not currently implemented)

The `iotd_Count` variable is important to this command (see `IOExtTD`). This variable contains the number of disk swaps made since the last booting procedure. One disk removal counts as one disk swap; one disk insertion counts as one disk swap. Therefore, removing one disk from a drive and inserting another counts as 2 disk swaps.

The following sequence opens the trackdisk device:

```
struct IOExtTD *DiskExtIO=0L;
#define TD_LEN (ULONG) (sizeof (struct IOExtTD))
...
   OpenDevice("trackdisk.device", Unit, &DiskExtIO, 0L, TD_LEN);
```

Remember that you must specify which drive you want to address when you open the device. The `Unit` parameter accepts the values 0 (for DF0:), 1 (for DF1:), 2 (for DF2:) or 3 (for DF3:). You can always open only one drive with `Open_A_Device()`. If you want to access multiple drives, you must call `Open_A_Device()` twice with two different device blocks and different values for `Unit`.

4.14.1 Reading and writing sectors

After `Open_A_Device()` you can begin. The following routine shows how to read a sector using the trackdisk device:

```
/*****************************************************************
*                  TrackDisk_ReadSector()       (Track_Support)*
* Function: Read sector                                          *
*----------------------------------------------------------------*
* Input - Parameter:                                             *
* DiskExtIO:    Device-Block                                     *
* SectorBuffer: Sector-Data                                      *
* LabelBuffer:  Label-Area data                                  *
* Offset:       Sector number                                    *
*****************************************************************/
VOID TrackDisk_ReadSector
(DiskExtIO,SectorBuffer,LabelBuffer,Offset)
struct IOExtTD          *DiskExtIO;
APTR                                SectorBuffer;
APTR                                        LabelBuffer;
ULONG                                              Offset;
{
```

```
   DiskExtIO->iotd_Count           = TrackDisk_GetDiskChangeCount
(DiskExtIO);
   DiskExtIO->iotd_Req.io_Offset   = Offset*512l;
   DiskExtIO->iotd_Req.io_Data     = (APTR) SectorBuffer;
   DiskExtIO->iotd_Req.io_Length   = (ULONG) TD_SECTOR;
   DiskExtIO->iotd_SecLabel        = (ULONG) LabelBuffer;
   Do_Command (DiskExtIO, (UWORD) ETD_READ);
}
```

This routine uses the `ETD_READ` command to read a sector from the disk. The number of disk swaps is saved in `iotd_Count` using `TrackDisk_GetDiskChangeCount()`. If the disk is swapped before the `ETD_READ` command, when the `ETD_READ` command is executed it determines that the value in `iotd_Count` is less than the value of the disk swap, and `ETD_READ` is not executed. Along with `ETD_READ` all of the ETD commands function after being checked by `iotd_Count`.

You can bypass this by saving the value 0xffffffff in `iotd_Count`. Here `iotd_Count` is always greater than or equal to the number of disk swaps. Use the `CMD_READ` command instead of `ETD_READ`, since `CMD_READ` does not disturb the `iotd_Count`.

To inform the trackdisk device which sector should be read, the offset of the sector to be read is given in `iotd_Req.io_Offset`. Notice that the trackdisk device numbers all of the sectors by byte. To read sector 0, the trackdisk device must be given the value 0. To read sector 1, the trackdisk device must be given the value 512 (a sector contains 512 bytes). To read sector 2, the trackdisk device must be given the value 1024, and so on. The device block performs the offset assignments, so you don't have to constantly multiply the sector number you want read by 512. All you need to do is specify the number of the sector to be read (0-1759).

Perhaps you want to read the lower portion of sector 0, or the top portion of sector 1. You can't read just parts of a sector—the offset must always be a factor of 512. The buffer in which you want to read the data of the sector should be a minimum size of 512 bytes, and should be in Chip memory. If you want to read more than one sector you must allocate more memory. The above routine reads only one sector. `SectorBuffer` contains the starting address of the data memory that must contain the 512 bytes.

If you enter a number for `SectorBuffer` less than 512 bytes, only the first 200 bytes are read. The other bytes stay protected from access. A sector on a disk also contains another data region beside the sector data—the label buffer. This 16-byte label buffer is placed before the actual buffer. Usually 0 bytes precede the actual buffer. You can use this label buffer as additional data memory, or write copyright messages in it.

You must provide a 16-byte label buffer for each sector to be read. This memory must be combined for multiple sector reading.

The following routine writes trackdisk data to a sector:

```
/***************************************************************
*                  TrackDisk_WriteSector()     (Track_Support)*
* Function: Write track                                       *
*-------------------------------------------------------------*
* Input - Parameter:                                          *
* DiskExtIO:    Device-Block                                  *
* SectorBuffer: Sector-Data (0x397c Bytes)                    *
* Offset:       Sector number                                 *
***************************************************************/
VOID TrackDisk_WriteSector
(DiskExtIO,SectorBuffer,LabelBuffer,Offset)
struct IOExtTD             *DiskExtIO;
APTR                                  SectorBuffer;
APTR                                               LabelBuffer;
ULONG
Offset;
{
   DiskExtIO->iotd_Count          = TrackDisk_GetDiskChangeCount
(DiskExtIO);
   DiskExtIO->iotd_Req.io_Offset  = Offset*512l;
   DiskExtIO->iotd_Req.io_Data    = (APTR) SectorBuffer;
   DiskExtIO->iotd_Req.io_Length  = (ULONG) TD_SECTOR;

   DiskExtIO->iotd_SecLabel       = (ULONG) LabelBuffer;

   Do_Command (DiskExtIO,(UWORD) ETD_WRITE);
   Do_Command (DiskExtIO,(UWORD) ETD_UPDATE);
}
```

This command sequence includes variables named `SectorBuffer` and `LabelBuffer`. This time the buffer contains the data that should be written to the disk. The offset is also given as with `TrackDisk_ReadSector()`. The main difference is that after the `WRITE` command an `UPDATE` command executes. This command ensures that the sector is physically written to the disk.

`ETD_WRITE` and `CMD_WRITE` ensure that the data written in the trackdisk device is the first written in a new track when accessed. This internal buffer contains enough memory to store an entire track (11 sectors). This buffer is usually the only one accessed with the `WRITE` and `READ` commands. Accesses through this buffer are very fast. If another track should be accessed, either the old trackbuffer is written again, or it reads a new track in the internal buffer. You can enlarge the internal buffer using `AddBuffers`.

Problems occur when you write a sector in the internal buffer by means of ETD_WRITE or CMD_WRITE, and then try to exit the program. The new data may not be written to the disk under certain conditions. This is why the UPDATE command executes after the WRITE command. This ensures that the internal buffer is written to the disk.

Knowing the current number of disk swaps is vital to the use of the extended commands. The following routine shows how the TD_CHANGENUM command finds this number:

```
/***************************************************************
*               TrackDisk_GetDiskChangeCount()    (Track_Support)*
* Function: Get number of disk changes                         *
*--------------------------------------------------------------*
* Input - Parameter:                                            *
* DiskExtIO: Device-Block                                       *
*--------------------------------------------------------------*
* Retrun value:                                                 *
* Number of disk changes                                        *
***************************************************************/
ULONG TrackDisk_GetDiskChangeCount(DiskExtIO)
struct IOExtTD                  *DiskExtIO;
{
   Do_Command (DiskExtIO,(UWORD)TD_CHANGENUM);
   return (DiskExtIO->iotd_Req.io_Actual);
}
```

After TD_CHANGENUM the actual number of disk swaps is stored in io_Actual.

We should also have control over the disk drive motor (on or off). The following routine controls the motor:

```
/***************************************************************
*                          TrackDisk_Motor()     (Track_Support)*
* Function: Motor on/off                                        *
*--------------------------------------------------------------*
* Input - Parameter:                                            *
* DiskExtIO: Device-Block                                       *
* Flag: TRUE  => Motor on                                       *
*       FALSE => Motor off                                      *
***************************************************************/
VOID TrackDisk_Motor (DiskExtIO,Flag)
struct IOExtTD       *DiskExtIO;
BOOL                           Flag;

{
   if (Flag)  DiskExtIO->iotd_Req.io_Length = 1l; /* Motor on */
   else       DiskExtIO->iotd_Req.io_Length = 0l; /* Motor off*/

   Do_Command (DiskExtIO,(UWORD) TD_MOTOR);
}
```

This command is important because the read and write commands turn the motor on but not off: The user must turn off the motor. The command TrackDisk_Motor (DiskExtIO, FALSE) writes a 0 in the length element of the `DiskExtIO` structure. `TrackDisk_Motor (DiskExtIO, TRUE)` writes a 1 into `io_Length`. This sets the motor high and addresses motor operation. You get the previous motor status from `io_Actual` (0=off, 1=on).

4.14.2 Reading and writing raw tracks

In addition to reading and writing of individual sectors, the trackdisk device also provides commands for reading and writing entire tracks. There is a small problem. Raw data doesn't appear in the usual byte format used by `ETD_READ` and `ETD_WRITE`.

The bits are encrypted (coded) before they are physically written to the disk. The Amiga uses MFM (Modified Frequency Modulation) coding. This means that the commands which read and write the track data are accessed physically, as if the data is really on the disk. Entire sync and clock bits are read consistently. This gives us the capability of reading data from foreign formats such as Atari.

You can also read disks that use the GCR (Group Code Recording) format. When writing you should specify which format the data should be written in. The following routines perform these read and write operations.

```
#define RAW_TRACK_LEN 0x397cl
/*****************************************************************
*                    TrackDisk_RawReadSector()  (Track_Support)*
*                                                               *
* Function: Read raw track (not processed!)                     *
*---------------------------------------------------------------*
* Input - Parameter:                                            *
*                                                               *
* DiskExtIO:   Device-Block                                     *
* TrackBuffer: Track-Data (0x397c Bytes)                        *
* Offset:      Track number                                     *
*****************************************************************/

VOID TrackDisk_RawReadSector (DiskExtIO,TrackBuffer,Offset)
struct IOExtTD              *DiskExtIO;
APTR                                   TrackBuffer;
ULONG                                              Offset;
{
   if (Offset > 159) DiskExtIO->iotd_Req.io_Error = 0xfc;
   else
   {
      DiskExtIO->iotd_Count            =
TrackDisk_GetDiskChangeCount (DiskExtIO);
      DiskExtIO->iotd_Req.io_Offset  = Offset;
      DiskExtIO->iotd_Req.io_Data    = (APTR) TrackBuffer;
      DiskExtIO->iotd_Req.io_Length  = RAW_TRACK_LEN;
      DiskExtIO->iotd_Req.io_Flags   = (BYTE) IOTDF_INDEXSYNC;
```

```
        DiskExtIO->iotd_Req.io_Actual =  0l;
        Do_Command (DiskExtIO,(UWORD) ETD_RAWREAD);
    }
}


/*****************************************************************
*                 TrackDisk_RawWriteSector()    (Track_Support)*
*                                                               *
* Function: Write raw track (unprocessed)                       *
*---------------------------------------------------------------*
* Input - Parameter:                                            *
*                                                               *
* DiskExtIO:   Device-Block                                     *
* TrackBuffer: Track-Data                                       *
* Offset:      Track number                                     *
*****************************************************************/

VOID TrackDisk_RawWriteSector (DiskExtIO,TrackBuffer,Offset)
struct IOExtTD                  *DiskExtIO;
APTR                                      TrackBuffer;
ULONG                                                Offset;
{
    if (Offset > 159) DiskExtIO->iotd_Req.io_Error = 0xfc;
    else
    {
        DiskExtIO->iotd_Count           =
TrackDisk_GetDiskChangeCount (DiskExtIO);
        DiskExtIO->iotd_Req.io_Offset  = Offset;
        DiskExtIO->iotd_Req.io_Data    = (APTR)  TrackBuffer;
        DiskExtIO->iotd_Req.io_Length  = (ULONG) RAW_TRACK_LEN;
        DiskExtIO->iotd_Req.io_Flags   = (BYTE) IOTDF_INDEXSYNC;
        Do_Command (DiskExtIO,(UWORD) ETD_RAWWRITE);
    }
}
```

Notice that a track now needs 0x397c bytes instead of 11*512 = 0x1600 bytes. This command should make it possible to wait for the exchange of the index locks and then begin reading and writing data. For this the `IOTDF_INDEXSYNC` flag is set. Unfortunately, a hardware error clears the hardware register before disk access. This makes the flag meaningless.

Because these commands do not leave the data format, only tracks 0-160 can be read and written. This is because the position of the read/write head is predetermined for each track. You don't need to multiply the sector number by 512 as with `ETD_READ` and `ETD_WRITE`. You simply give the number of the track to be read.

4.14.3 Formatting a disk

You've seen the most frequently used trackdisk commands. However, you can still do more with this device. For example, you have the option of formatting individual tracks. This is useful, for example, when the boot block on a disk was destroyed for some reason and so DOS cannot access the disk. Assuming that there is no important data on track 0 (sectors 0 through 10), you can format track 0 and copy the boot block of an intact disk to the disk whose boot block you reformatted, by means of `READ` and `WRITE`. Then you can rescue the most important files to the new disk, assuming the rest of the disk is in order. The following routine shows how a single track can be formatted:

```
#define MEMTYPE (ULONG) MEMF_CLEAR|MEMF_CHIP
/****************************************************************
*                        TrackDisk_Format()      (Track_Support)*
*                                                                *
* Function: Format track                                         *
*----------------------------------------------------------------*
* Input - Parameter:                                             *
*                                                                *
* DiskExtIO: Device-Block                                        *
* Offset:    Track number                                        *
****************************************************************/

VOID TrackDisk_Format (DiskExtIO,Offset)
struct IOExtTD        *DiskExtIO;
ULONG                          Offset;
{
   BYTE *FormatData=0l;
   UWORD i;

   FormatData = (BYTE *) AllocMem (NUMSECS*TD_SECTOR, MEMTYPE);
   if (FormatData == 0l)
      CloseIt("No FormatData Buffer !!!");

   for (i=0; i<NUMSECS*TD_SECTOR; i+= 4)
   {
      FormatData[i]   = ' ';
      FormatData[i+1] = 'd';  /* data   */
      FormatData[i+2] = 'b';  /* becker */
      FormatData[i+3] = ' ';
   }

   DiskExtIO->iotd_Count          = TrackDisk_GetDiskChangeCount
(DiskExtIO);
   DiskExtIO->iotd_Req.io_Data    = (APTR) FormatData;
   DiskExtIO->iotd_Req.io_Length  = (ULONG) (NUMSECS*TD_SECTOR);
   DiskExtIO->iotd_Req.io_Offset  = Offset;
   Do_Command (DiskExtIO, (UWORD)ETD_FORMAT);
   FreeMem (FormatData,NUMSECS*TD_SECTOR);
}
```

Unlike READ and WRITE, the number of the tracks to be formatted (0-160) is given in io_Offset. Also, a multiplication by 512 is unnecessary. Memory for an entire track is reserved for this command and it is written with db. Then the format buffer is given in the io_Data pointer. Be aware that you can only format one or more complete tracks with TD_FORMAT and ETD_FORMAT, with one of the tracks to format the size of the format buffer NUMSECS*TD_SECTOR is (11*512). The memory size adjusts accordingly with multiple tracks to be formatted.

Tracks and sectors are described according to a certain pattern when formatting. Whoever has the time and the inclination can look at a newly formatted disk with the disk monitor listed in this chapter and analyze the formatting pattern.

4.14.4 Status commands

A number of trackdisk commands indicate the status of the disk and disk drive. The following routine demonstrates this command set:

```
/***************************************************************
*                    TrackDisk_GetProtStatus()  (Track_Support)*
* Function: Write-Protect on/off ?                             *
*--------------------------------------------------------------*
* Input - Parameter:                                           *
* DiskExtIO: Device-Block                                      *
*--------------------------------------------------------------*
* Return value:                                                *
* 0: not write protected   <>0: write protected                *
***************************************************************/
ULONG TrackDisk_GetProtStatus (DiskExtIO)
struct IOExtTD      *DiskExtIO;
{
   Do_Command (DiskExtIO,(UWORD) TD_PROTSTATUS);
   return (DiskExtIO->iotd_Req.io_Actual);
}
/***************************************************************
*                 TrackDisk_GetChangeState()    (Track_Support)*
* Function: Is disk inserted?                                  *
*--------------------------------------------------------------*
* Input - Parameter:                                           *
* DiskExtIO: Device-Block                                      *
*--------------------------------------------------------------*
* Return value:                                                *
* 0: Diskette inserted   <>0: Diskette removed                 *
***************************************************************/
ULONG TrackDisk_GetChangeState (DiskExtIO)
struct IOExtTD                  *DiskExtIO;
{
   Do_Command (DiskExtIO,(UWORD) TD_CHANGESTATE);
   return (DiskExtIO->iotd_Req.io_Actual);
}
```

TD_PROSTATUS places the write protect status in io_Actual. If io_Actual equals zero, the disk is not write protected. TD_CHANGESTATE operates in a similar manner. If io_Actual equals zero following this command, there is no disk in the disk drive. The following commands also place return values in io_Actual:

TD_GETDRIVETYPE

This command helps you determine what disk drive should be accessed. If io_Actual equals 1, a 3-1/2" drive is connected. If io_Actual equals 2, a 5-1/2" drive is connected.

TD_GETNUMTRACKS

This command returns the number of tracks that the connected disk drive can handle. The standard 3-1/2" Amiga disk drives manage 160 tracks.

TD_SEEK
ETD_SEEK

This command lets you set and test the read/write head on the given track to see if it is positioned at a certain track. You supply the number of the first sector of the given track (factor of 11). If the track was incorrect, this quits with an error (io_Error !=0):

```
/***************************************************************
*                        TrackDisk_SeekSector()  (Track_Support)*
*                                                              *
* Function: Position read head                                 *
*--------------------------------------------------------------*
* Input - Parameter:                                           *
*                                                              *
* DiskExtIO: Device-Block                                      *
* Offset:    Sector number                                     *
***************************************************************/

VOID TrackDisk_SeekSector (DiskExtIO,Offset)
struct IOExtTD          *DiskExtIO;
ULONG                              Offset;
{
   DiskExtIO->iotd_Req.io_Offset  = Offset*512l;
   Do_Command (DiskExtIO,(UWORD) TD_SEEK);
}
```

Should this command return an error, this may mean that your disk drive is damaged.

4.14.5 Disk interrupts

Another interesting feature of the trackdisk device is the possibility of installing an interrupt that executes with each disk swap. The `TD_REMOVE` command performs this task.

```
struct Interrupt *Interrupt = 0l;
...
/****************************************************************
*                    TrackDisk_InterruptOn()    (Track_Support)*
*                                                              *
* Function: Install disk change interrupt                      *
*--------------------------------------------------------------*
* Input - Parameter:                                           *
*                                                              *
* DiskExtIO:            Device-Block                           *
* TrackDisk_DiskRemove: Interrupt-Routine                      *
****************************************************************/
VOID TrackDisk_InterruptOn (DiskExtIO,TrackDisk_DiskRemoved)
struct IOExtTD              *DiskExtIO;
VOID                                  (*TrackDisk_DiskRemoved)();
{
   Interrupt = (struct Interrupt *) AllocMem
((ULONG)(sizeof(struct Interrupt)),MEMTYPE);
   if (Interrupt == 0l)  CloseIt ("NoInterrupt");
   Interrupt->is_Code            = (VOID (*)())
TrackDisk_DiskRemoved;
   DiskExtIO->iotd_Req.io_Data    = (APTR)  Interrupt;
   DiskExtIO->iotd_Req.io_Command = (UWORD) TD_REMOVE;
   DoIO (DiskExtIO);
}
```

This routine allocates memory for an interrupt. The name `interrupt` may not be used in your program if you want to use this routine. The interrupt structure stores the address of your interrupt routine in the `is_Code` pointer. The interrupt structure is not assigned a value in the `is_Data` array. It is erased from the memory allocation (`MEMF_CLEAR`). Should you give a memory region here, this is given to the interrupt routine in A1.

After interrupt structure initialization, your address is given in the `io_Data` pointer of the device block and `TD_REMOVE` is called. Now the interrupt is installed. To release the interrupt again, you can use the following routine:

```
/****************************************************************
*                  TrackDisk_InterruptOff()    (Track_Support)*
* Function: Remove Disk-Interrupt                              *
*--------------------------------------------------------------*
* Input - Parameter:                                           *
* DiskExtIO: Device-Block                                      *
****************************************************************/
```

```
VOID TrackDisk_InterruptOff (DiskExtIO)
struct IOExtTD              *DiskExtIO;
{
   DiskExtIO->iotd_Req.io_Data    = 0l;
   DiskExtIO->iotd_Req.io_Command = (UWORD) TD_REMOVE;
   DoIO (DiskExtIO);
   if (Interrupt != 0l)
      FreeMem (Interrupt,(ULONG)(sizeof (struct Interrupt)));
   Interrupt = (struct Interrupt *) 0l; /* f|r CloseIt() */
}
```

This routine sets the `io_Data` pointer (which points to the interrupt structure) to 0 and calls `TD_REMOVE` a second time. Then the program releases the interrupt structure's memory. This is why `interrupt` is globally defined and may not be used in your programs. `TrackDisk_InterruptOn()` and `TrackDisk_InterruptOff()` access `interrupt`.

In addition to `TD_REMOVE`, KICK1.2 includes some disk interrupt processing commands: `TD_ADDCHANGEINT` and `TD_REMCHANGEINT`. You can install an interrupt with `TD_ADDCHANGEINT` just like you did with `TD_REMOVE`:

```
/*************************************************************
*                  TrackDisk_AddChangeInt()    (Track_Support)*
* Function: Disk-Interrupt install                            *
*          Attention: TD_REMCHANGEINT does not function!!!!   *
*-------------------------------------------------------------*
* Input - Parameter:                                          *
* DiskExtIO:              Device-Block                        *
* TrackDIsk_DiskRemoved: Interrupt-Routine                    *
*************************************************************/
VOID TrackDisk_AddChangeInt (DiskExtIO,TrackDisk_DiskRemoved)
struct IOExtTD              *DiskExtIO;
VOID
(*TrackDisk_DiskRemoved)();
{
   InterruptIO = (struct IOExtTD *)GetDeviceBlock (TD_LEN);
   TrackDisk_Copy (DiskExtIO,InterruptIO);
   Interrupt = (struct Interrupt *)
      AllocMem ((ULONG)(sizeof(struct Interrupt)),MEMTYPE);
   if (Interrupt == 0l)  CloseIt ("NoInterrupt");
   Interrupt->is_Code            = (VOID (*) ())
TrackDisk_DiskRemoved;
   InterruptIO->iotd_Req.io_Data    = (APTR)  Interrupt;
   InterruptIO->iotd_Req.io_Command = (UWORD) TD_ADDCHANGEINT;
   SendIO (InterruptIO);
}
```

Memory for the interrupt structure is allocated in this routine. Another device block is also constructed, because the `TD_ADDCHANGEINT` command returns to the program when the interrupt is released. Since you want to work with the trackdisk device in the meantime, a second device block must be added. The most important variables are copied into this one by means of `TrackDisk_Copy()`:

```
/****************************************************************
*                       TrackDisk_Copy()         (Track_Support)*
*                                                               *
* Function: Device-Block copy                                   *
*---------------------------------------------------------------*
* Input - Parameter:                                            *
*                                                               *
* OldExtIO: Original                                            *
* NewExtIO: Copy                                                *
****************************************************************/
VOID TrackDisk_Copy (OldExtIO, NewExtIO)
struct IOExtTD      *OldExtIO,*NewExtIO;
{
   NewExtIO->iotd_Req.io_Device =
      OldExtIO->iotd_Req.io_Device;
   NewExtIO->iotd_Req.io_Unit =
      OldExtIO->iotd_Req.io_Unit;
   NewExtIO->iotd_Count =
      OldExtIO->iotd_Count;
}
```

Unfortunately there is a problem with using `TD_ADDCHANGEINT` or `TD_REMCHANGEINT`. The interrupts must be removed before closing the trackdisk device. The command provided to do this is `TD_REMCHANGEINT`, but this command is not currently implemented. We recommend that you install a disk interrupt using `TrackDisk_AddChangeInt()` because you can never remove these, and you can run into big trouble with these after the trackdisk device closes.

4.14.6 Error handling

Many inexperienced trackdisk programmers encounter program errors. We recommend that you use our track support routines to minimize problems. In addition to the normal device errors, the trackdisk device has device specific errors:

TDERR_NotSpecified	(20)	error could not be determined
TDERR_NoSecHdr	(21)	sector header not found
TDERR_BadSecPreamble	(22)	error in sector preamble
TDERR_BadSecID	(23)	error in sector identifier
TDERR_BadHdrSumm	(24)	checksum error in header
TDERR_BadSecSum	(25)	checksum error in sector
TDERR_TooFewSecs	(26)	too few or too many sectors on track
TDERR_BadSecHdr	(27)	sector header unreadable
TDERR_WriteProt	(28)	disk write protected
TDERR_DiskChanged	(29)	no disk inserted or disk changed
TDERR_SeekError	(30)	seek error during seek position verification

TDERR_NoMem	(31)	insufficient memory
TDERR_BadUnitNum	(32)	drive addressed not connected
TDERR_BadDriveType	(33)	incompatible disk drive
TDERR_DriveInUse	(34)	drive already in use
TDERR_PostReset	(35)	user hit reset, waiting for reboot

The following routine returns either the error number (for a device unspecified error), or the error in text format:

```
struct StrPack
{
   BYTE  *String;
   ULONG  Len;
};

BYTE              HTab[] = {'0','1','2','3','4','5','6','7',
                            '8','9','A','B','C','D','E','F'};
BYTE *ErrorStrings[] = {"Not Specified\012\015",
                        "No Sector Header\012\015",
                        "Bad Sector Preamble\012\015",
                        "Bad Sector ID\012\015",
                        "Bad Header Sum\012\015",
                        "Bad Sector Sum\012\015",
                        "Too Few Sectors\012\015",
                        "Bad Sector Header\012\015",
                        "Write Protected\012\015",
                        "Disk Changed\012\015",
                        "Seek Error\012\015",
                        "Not enough memory\012\015",
                        "Bad Unit Number\012\015",
                        "Bad Drive Type\012\015",
                        "Drive In Use\012\015",
                        "Post Reset\012\015"};

/****************************************************************
*                    TrackDisk_ProcessError()    (Track_Support)*
* Function: Processes Trackdisk-Error                          *
*--------------------------------------------------------------*
* Input - Parameter:                                           *
* DiskExtIO: Device-Block                                      *
* StrPack: String-Packet for Error-String                      *
****************************************************************/
VOID TrackDisk_ProcessError (DiskExtIO,StrPack)
struct IOExtTD            *DiskExtIO;
struct StrPack                      *StrPack;
{
          BYTE  Error;
   static BYTE  *ErrStr;
   Error = DiskExtIO->iotd_Req.io_Error;
   StrPack->String = 0l;
   StrPack->Len    = 0l;
   if (Error != (BYTE) 0)
   {
      if ((Error >= (BYTE) 20) && (Error <= (BYTE) 35))
      {
         StrPack->String = ErrorStrings[(Error-(BYTE)20)];
         StrPack->Len    = (ULONG) strlen (StrPack->String);
      }
      else
```

```
        {
           ErrStr      = "\012\015Error #  \012\015";
           ErrStr[9]   = HTab[(Error>>4)&15];
           ErrStr[10]  = HTab[Error&15];
           StrPack->String = ErrStr;
           StrPack->Len    = 131;
        }
    }
}
```

When you call this routine you must give the address of a string packet as a parameter. Either the text of the error or a string in the format `Error #nn` is given in this packet. You can then display this string.

This routine must be called directly after executing `READ` or `WRITE` (`TD_MOTOR` is usually executed without an error). There is no problem with the routine `TrackDisk_WriteSector()` because the `UPDATE` command executes after `WRITE`. Here, as the user, you must either call `UPDATE` after `ProcessError()` or integrate `ProcessError()` into `WriteSector()`.

Combine the listed Track_Support files to form the Track_Support.c file. The file header appears as follows:

```
/****************************************************************
*                       Track_Support.c                         *
*                        August 1988                            *
*                     (c) Bruno Jennrich                        *
* Compile-Info:                                                 *
* cc Track_Support.c                                            *
****************************************************************/
#include "exec/types.h"
#include "exec/memory.h"
#include "exec/devices.h"
#include "exec/interrupts.h"
#include "devices/trackdisk.h"


#define RAW_TRACK_LEN 0x397cl

#define MEMTYPE (ULONG) MEMF_CLEAR|MEMF_CHIP

#define TD_LEN (ULONG) (sizeof (struct IOExtTD))

struct IOExtTD *InterruptIO = 0l;
struct Interrupt *Interrupt = 0l;

VOID *AllocMem();
VOID *GetDeviceBlock();
```

4.14.7 The disk editor

The following program allows you to read and write sectors, format tracks and read disk status. You also have the option of writing the sector contents to a file.

```
/*****************************************************************
*                           DiskEd.c                             *
*                      (c) Bruno Jennrich                        *
* Compile-Info:                                                  *
* cc DiskEd.c                                                    *
* ln DiskEd.o Track_Support.o Con_Support.o Devs_Support.o -lc *
*****************************************************************/
#include "exec/types.h"
#include "exec/memory.h"
#include "exec/devices.h"
#include "exec/interrupts.h"
#include "graphics/gfxbase.h"
#include "libraries/dos.h"
#include "devices/trackdisk.h"
#include "devices/console.h"
#include "devices/keymap.h"
#include "intuition/intuitionbase.h"
#include "intuition/intuition.h"
#define MEMTYPE    (MEMF_CHIP | MEMF_CLEAR)
#define CON_LEN (ULONG) (sizeof (struct IOStdReq))
#define TD_LEN (ULONG) (sizeof (struct IOExtTD))
#define RAW_TRACK_LEN   0x397cl
VOID *CreatePort();
VOID *CreateExtIO();
VOID *Open();
VOID *AllocMem();
VOID *GetDeviceBlock();
VOID *OpenLibrary();
VOID *OpenScreen();
VOID *OpenWindow();
extern VOID TrackDisk_DiskRemoved();
struct IntuitionBase *IntuitionBase = 0l;
struct Screen        *Screen        = 0l;
struct Window        *Window        = 0l;
struct NewScreen      NewScreen = {
                                     0,0,640,200,4,
                                     0,1,
                                     HIRES,
                                     CUSTOMSCREEN,
                                     0l,
                                     (UBYTE*) "No Name",
                                     0l,
                                     0l
                                  };
struct NewWindow      NewWindow = {
                  0,0,
                  640,200,
                  0,1,
                  0l,
                  (ULONG) ACTIVATE,
```

```
                    01,
                    01,
                    (UBYTE*) "TrackDisk-Editor (c) Bruno Jennrich",
                    01,
                    01,
                    0,0,
                    0,0,
                    CUSTOMSCREEN
                    };
extern struct Interrupt *Interrupt;
extern struct IOExtTD   *InterruptIO;
struct IOExtTD   *DiskExtIO   = 01;
struct IOStdReq  *ConsoleRead  = 01,
                 *ConsoleWrite = 01;
UWORD            *File         = 01;
BYTE             *SectorBuffer = 01;
BYTE             *TrackBuffer  = 01;
BYTE             *LabelBuffer  = 01;
BYTE              ReadBuffer[256];   /* Keyboard-Buffer */
BYTE              HexTab[] = {'0','1','2','3','4','5','6','7',
                              '8','9','A','B','C','D','E','F'};
BYTE              ConversionTable[256];
struct StrPack
{
   BYTE  *String;
   ULONG  Len;
};
struct Offset
{
   ULONG Track;
   ULONG Sector;
   ULONG Side;
};
/*****************************************************************
*                        CloseIt()                        (User)*
* Function: In case of erro close everything                    *
* Input - Parameter:                                            *
* String: Error-Message                                         *
*****************************************************************/
VOID CloseIt (String)
char         *String;
{
   UWORD i;
   UWORD *dff180 = (UWORD *)0xdff180;
   UWORD Error = 0;
   if (strlen (String) > 01)
   {
      for (i=0;i<0xffff;i++)  *dff180 = i;
      puts (String);
      Error = 10;
   }
   if (Window != 01)        CloseWindow (Window);
   if (Screen != 01)        CloseScreen (Screen);
   if (IntuitionBase != 01) CloseLibrary (IntuitionBase);
   if (Interrupt != 01)     TrackDisk_InterruptOff(DiskExtIO);
   if (DiskExtIO->iotd_Req.io_Device != -11) Close_A_Device
(DiskExtIO);
   else FreeDeviceBlock (DiskExtIO);
   if (SectorBuffer != 01)
      FreeMem (SectorBuffer,TD_SECTOR);
   if (TrackBuffer != 01)
      FreeMem (TrackBuffer, RAW_TRACK_LEN);
```

```
   if (LabelBuffer != 0l)
      FreeMem (LabelBuffer, (ULONG)(NUMSECS));
   if (ConsoleRead != 0)    Close_A_Device (ConsoleRead);
   if (ConsoleWrite != 0)   FreeDeviceBlock (ConsoleWrite);
   if (File != 0l)          Close (File);
   exit (Error);
}
/****************************************************************
*                    TrackDisk_DiskRemoved()           (User)*
* Function: Interrupt-Routine called by disk change          *
****************************************************************/
#asm
   public   _TrackDisk_DiskRemoved
_TrackDisk_DiskRemoved:
   move.w #$ffff,d0
  loopa:
   move.w   d0,$dff180
   dbra     d0,loopa
   rts
#endasm
/****************************************************************
*                          ReadCommand()             (User)*
* Function: Read keyboard input (Console-Device)              *
* Input - Parameter:                                          *
* Buffer: Go to where with the input?                         *
* MaxLength: Maximum number of key presses                    *
****************************************************************/
VOID ReadCommand (Buffer,MaxLength)
BYTE              *Buffer;
ULONG                     MaxLength;
{
   BYTE *BufPointer;
   BYTE  Character;
   ULONG Length;

   Character = (BYTE) 0;
   Length = (ULONG)0;
   BufPointer = Buffer;
   while (Character != (BYTE)13)
   {                                               /* <RETURN> */
      Console_Read  (ConsoleRead, &Character,1l);
      if (Character == (BYTE)8)   /* Backspace */
      {
         if (Length > 0l)
         {
            *BufPointer-- = (BYTE) '\000';  /* mark String end*/
            Console_Write (ConsoleWrite,&Character,1l);
                                                /* Backspace */
            Console_Write (ConsoleWrite," ",1l); /*Delete char*/
            Console_Write (ConsoleWrite,&Character,1l);
                                                /* Backspace */
            Length--;
         }
      }
      else
      if (Length < MaxLength)
      {
         if (Character == (BYTE)13) Console_Write
(ConsoleWrite,"\012",1l);
         if (((Character | (BYTE) 0x20) >= 'a') &&
             ((Character | (BYTE) 0x20) <= 'z'))
            *BufPointer++ = (Character | (BYTE) 0x20);
```

```
            else
                *BufPointer++ = Character;
            Length++;
            Console_Write (ConsoleWrite,&Character,1l);
        }
    }
    *BufPointer = (BYTE)0;
}
/***************************************************************
*                      HEXAtoULONG()                     (User)*
* Function: Convert Hex-String to ULONG                        *
* Input - Parameter:                                           *
* BufPointer: Addess of the Hex-String                         *
* Return value:                                                *
* Valuse of the Hex-Strings ("ABC" = 0xabc)                    *
***************************************************************/
ULONG HEXAtoULONG (BufPointer)
BYTE          *BufPointer;
{
    UWORD i;
    ULONG Val = 0l;
    UWORD Len;
    Len = strlen (BufPointer);
    for (i=0;i<Len;i++)
    {
        if ((*BufPointer>= 'A') && (*BufPointer <= 'F'))
        {
            Val *= 16l;
            Val += ((*BufPointer && (0xff-0x20))-'A'+(BYTE)10);
        }
        else
        if ((*BufPointer>= '0') && (*BufPointer <= '9'))
        {
            Val *= 16l;
            Val += (ULONG)(*BufPointer-'0');
        }
        BufPointer++;
    }
    return (Val);
}
/***************************************************************
*                      DEZAtoULONG()                     (User)*
* Function: Convert Decimal-String to ULONG                    *
* Input - Parameter:                                           *
* BufPointer: Address of the Decimal-Strings                   *
* Return value:                                                *
* Value of the Decimal-Strings ("123" = 123)                   *
***************************************************************/

ULONG DEZAtoULONG (BufPointer)
BYTE          *BufPointer;
{
    UWORD i;
    ULONG Val = 0l;
    UWORD Len;
    Len = strlen (BufPointer);
    for (i=0;i<Len;i++)
    {
        if ((*BufPointer>= '0') && (*BufPointer <= '9'))
        {
            Val *= 10l;
            Val += (ULONG)(*BufPointer-'0');
```

```
         BufPointer++;
      }
   }
   return (Val);
}
/****************************************************************
*                          LONGtoA()                    (User)*
* Function: Convert LONG value to ASCII and display           *
* Input - Parameter:                                           *
* Val: LONG value                                              *
****************************************************************/
VOID LONGtoA (Val)
ULONG         Val;
{
   ULONG Start = 10000000001;
   BYTE  Ascii[11];
   UWORD i;
   i=0;
   do
   {
      Ascii[i] = '0';
      while (Val >= Start)
      {
         Val -= Start;
         Ascii[i]++;
      }
      i++;
      Start /= 101;
   }
   while (Start != 01);
   Console_Write (ConsoleWrite,Ascii,101);
}
/****************************************************************
*                         UWORDtoHex()                  (User)*
* Function: Convert UWORD value to Hex-String                  *
* Input - Parameter:                                           *
* Val: UWORD value                                             *
* Buffer: where with Hex-Strings                               *
****************************************************************/
VOID UWORDtoHex (Val,Buffer)
UWORD           *Val;
BYTE                *Buffer;
{
   UWORD Hex;
   Hex       = *Val & 0xf000;
   Hex       = Hex >> 12;
   Buffer[0] = HexTab[Hex];
   Hex       = *Val & 0x0f00;
   Hex       = Hex >> 8;
   Buffer[1] = HexTab[Hex];
   Hex       = *Val & 0x00f0;
   Hex       = Hex >> 4;
   Buffer[2] = HexTab[Hex];
   Hex       = *Val & 0x000f;
   Buffer[3] = HexTab[Hex];
}
/****************************************************************
*                          Display()                    (User)*
* Function: Display sector contents                            *
* Input - Parameter:                                           *
* Offset: Sector number                                        *
****************************************************************/
```

```
VOID Display (SectorBuffer,LabelBuffer,Offset)
BYTE         *SectorBuffer;
BYTE                     *LabelBuffer;
ULONG                                Offset;
{
   UWORD i,j;
   UWORD BufPos = 0;
   BYTE String[20];
   BYTE HexBuffer[72*16];
   BYTE AsciiBuffer[72*8];
   BYTE LabelBuff[8*5];
   BYTE LabelAscii[17];
   BYTE OffsetBuffer[14];
   UWORD Offs;
   OffsetBuffer[ 0] = 'B';
   OffsetBuffer[ 1] = 'l';
   OffsetBuffer[ 2] = 'o';
   OffsetBuffer[ 3] = 'c';
   OffsetBuffer[ 4] = 'k';
   OffsetBuffer[ 5] = ':';
   OffsetBuffer[ 6] = ' ';
   OffsetBuffer[ 7] = '$';
   OffsetBuffer[ 8] = ' ';
   OffsetBuffer[ 9] = ' ';
   OffsetBuffer[10] = ' ';
   OffsetBuffer[11] = ' ';
   OffsetBuffer[12] = '\012';
   OffsetBuffer[13] = '\015';
   Offs = (UWORD) Offset;/* convert to UWORD for UWORDtoHex() */
   UWORDtoHex (&Offs,&OffsetBuffer[8]);
   Console_Write (ConsoleWrite,OffsetBuffer,14l);
   Console_Write (ConsoleWrite,"Labelbuffer:\012\015",-1l);
   for (i=0;i<8;i++)
   {
      LabelBuff [i*5] = ' ';
      UWORDtoHex ((LabelBuffer+i*2),&LabelBuff[i*5+1]);
   }
   Console_Write (ConsoleWrite,LabelBuff,8l*5l);
   Console_Write (ConsoleWrite,"\012\015",-1l);
   LabelAscii[0] = ' ';
   for (i=0;i<16;i++)
   {
      LabelAscii [i+1] =
ConversionTable[(UBYTE)*(LabelBuffer+i)];
   }
   Console_Write (ConsoleWrite,LabelAscii,17l);
   Console_Write (ConsoleWrite,"\012\015\012\015",-1l);
   for (i=0;i<16;i++)       /* 16 Lines */
   {
      HexBuffer[i*72] = ' ';
      UWORDtoHex(&BufPos,&HexBuffer[i*72+1]);
      HexBuffer[i*72+5] = ' ';
      for (j=0;j<16;j++)/* 32 Bytes (16 UWORDS)=64 characters */
      {
         UWORDtoHex
(&SectorBuffer[BufPos],&HexBuffer[i*72+6+j*4]);
         BufPos += 2;
      }
      HexBuffer[i*72+70] = '\012';
      HexBuffer[i*72+71] = '\015';
   }
   Console_Write (ConsoleWrite,HexBuffer,16l*72l);
```

```
   BufPos = 0;
   for (i=0;i<8;i++)
   {
      AsciiBuffer[i*72] = ' ';
      UWORDtoHex(&BufPos,&AsciiBuffer[i*72+1]);
      AsciiBuffer[i*72+5] = ' ';
      for (j=0;j<64;j++)
      {
         AsciiBuffer[i*72+6+j] =
ConversionTable[(UBYTE)SectorBuffer[i*64+j]];
         BufPos ++;
      }
      AsciiBuffer[i*72+70] = '\012';
      AsciiBuffer[i*72+71] = '\015';
   }
   Console_Write (ConsoleWrite,AsciiBuffer,81*72l);
   if (File != 0)
   {
      Write (File,"\012",-1l);
      Write (File,OffsetBuffer,13l);
      Write (File,"Labelbuffer: \012",14l);
      Write (File,LabelBuff,81*5l);
      Write (File,"\012",1l);
      Write (File,LabelAscii,17l);
      Write (File,"\012-------------------------------\012",73l);
      for (i=0;i<16;i++)
      {
         HexBuffer[i*72+70] = ' ';
         HexBuffer[i*72+71] = '\012';
      }
      for (i=0;i<8;i++)
      {
         AsciiBuffer[i*72+70] = ' ';
         AsciiBuffer[i*72+71] = '\012';
      }
      Write (File,HexBuffer,(72l*16l));
      Write (File,"\012",1l);
      Write (File,AsciiBuffer,(72l*8l));
   }
}
/*****************************************************************
*                        GetOffset()                      (User)*
* Function: Get offset for READ/WRITE from input string         *
* Input - Parameter:                                            *
* BufPointer: Addesseof the Input - Strings (HEX or DEC)        *
* Return value:                                                 *
* Offset recieved                                               *
*****************************************************************/
ULONG GetOffset (BufPointer)
BYTE          *BufPointer;
{
   BYTE *SecBuf;
   SecBuf = BufPointer;
   while ((*BufPointer != (BYTE)0) && (*BufPointer != '$') &&
(*BufPointer != '#')) BufPointer++;
   if (*BufPointer == (BYTE) 0) return (-1l);
   else
   if (*BufPointer == '$') return(HEXAtoULONG(BufPointer+1));
   else
   if (*BufPointer == '#') return(DEZAtoULONG(BufPointer+1));
   else
   if ((*BufPointer >= '0') && (*BufPointer <= ' 9'))
```

```
    return(DEZAtoULONG (SecBuf));
}
/*****************************************************************
*                    HandleCommands()                   (User)*
* Function: Process commands entered                          *
*****************************************************************/
VOID HandleCommands ()
{
   BYTE  *BufPointer;
   BYTE   Command;
   BOOL   QuitFlag;
   struct StrPack StrPack;
   ULONG  Count;
   ULONG Offset = 0l;
   QuitFlag = FALSE;
   while (!QuitFlag)
   {
      ReadCommand (ReadBuffer,256l);
      BufPointer = ReadBuffer;
      while (*BufPointer == ' ') BufPointer++;
      /* skip spaces */
      Command = *BufPointer;
      /* the first character after ' ' is the commnand */
      switch (Command)  /* Which command? */
      {
         case (BYTE)'h':
            /* help */
            Console_Write (ConsoleWrite,"\012\015r#/$[Block]
- Read Sector\012\015",43l);
            Console_Write (ConsoleWrite,"w#/$[Block]
- Write Sector\012\015",41l);
            Console_Write (ConsoleWrite,"f[Track]
- Format Track\012\015",41l);
            Console_Write (ConsoleWrite,"s
- Disk Status\012\015",40l);
            Console_Write (ConsoleWrite,"d
- Display Sector\012\015",43l);
            Console_Write (ConsoleWrite,"q
- Quit\012\015",34l);
            Console_Write (ConsoleWrite,"h
- This Reference\012\015",43l);
         break;
         case (BYTE)'r':
            /* Read */
            Offset = GetOffset (BufPointer);
            TrackDisk_Motor(DiskExtIO,TRUE);
            TrackDisk_ReadSector
(DiskExtIO,SectorBuffer,LabelBuffer,Offset);
            TrackDisk_ProcessError (DiskExtIO,&StrPack);
            TrackDisk_Motor(DiskExtIO,FALSE);
            Console_Write
(ConsoleWrite,StrPack.String,StrPack.Len);
         break;
         case (BYTE)'w':
            /* Write */
            Offset = GetOffset (BufPointer);
            TrackDisk_Motor(DiskExtIO,TRUE);
            TrackDisk_WriteSector
(DiskExtIO,SectorBuffer,LabelBuffer,Offset);
            TrackDisk_ProcessError (DiskExtIO,&StrPack);
            TrackDisk_Motor(DiskExtIO,FALSE);
```

```
            Console_Write
(ConsoleWrite,StrPack.String,StrPack.Len);
         break;
         case (BYTE)'f':
            Offset = GetOffset (BufPointer);
            TrackDisk_Motor(DiskExtIO,TRUE);
            TrackDisk_Format (DiskExtIO,Offset);
            TrackDisk_ProcessError (DiskExtIO,&StrPack);
            TrackDisk_Motor(DiskExtIO,FALSE);
            Console_Write
(ConsoleWrite,StrPack.String,StrPack.Len);
         break;
         case (BYTE)'s':
            Console_Write (ConsoleWrite,"\012\015DiskChangeCount
: ",-1l);
            DiskExtIO->iotd_Count =
TrackDisk_GetDiskChangeCount(DiskExtIO);
            LONGtoA (DiskExtIO->iotd_Count);
            Console_Write (ConsoleWrite,"\012\015",-1l);
            if (TrackDisk_GetProtStatus(DiskExtIO) != 0l)
               Console_Write (ConsoleWrite,"Disk
protected\012\015",-1l);
            else
               Console_Write (ConsoleWrite,"Disk not
protected\012\015",-1l);
            if (TrackDisk_GetChangeState(DiskExtIO) == 0l)
               Console_Write (ConsoleWrite,"Disk
inserted\012\015",-1l);
            else
               Console_Write (ConsoleWrite,"Disk
removed\012\015",-1l);
            /* Status */
         break;
         case (BYTE)'d':
            /* Display */
            Display(SectorBuffer,LabelBuffer,Offset);
         break;
         case (BYTE)'q':
            QuitFlag = (BOOL)TRUE;
         break;
         case (BYTE)13:
            /* Intercept Return  */
         break;
         default:
            Console_Write (ConsoleWrite,"\012\015\Bad
Command!!!\012\015",-1l);
         break;
      }
   }
}
/***************************************************************
*                 The_TrackDisk_Device()              (User)*
* Function: Use Trackdisk-Device                             *
* Input - Parameter:                                         *
* Unit: Which disk drive ?                                   *
***************************************************************/
VOID The_Trackdisk_Device (Unit)
ULONG                      Unit;
{
   ULONG Offset;
   Open_A_Device ("trackdisk.device",Unit,&DiskExtIO,0l,TD_LEN);
   TrackDisk_InterruptOn(DiskExtIO,TrackDisk_DiskRemoved);
```

```
   HandleCommands ();
   TrackDisk_InterruptOff(DiskExtIO);
   Close_A_Device (DiskExtIO);
}
/************************************************************
*                     Open_Screen_and_Window()        (User)*
* Function: Editor Screen and Window open                   *
************************************************************/
VOID Open_Screen_and_Window()
{
   IntuitionBase = (struct IntuitionBase*)
                   OpenLibrary ("intuition.library",0l);
   if (IntuitionBase == 0l) CloseIt ("No IntuitionBase !");
   Screen = (struct Screen *) OpenScreen (&NewScreen);
   if (Screen == 0l) CloseIt ("No Screen !");
   NewWindow.Screen = Screen;
   Window = (struct Window *) OpenWindow (&NewWindow);
   if (Window == 0l) CloseIt ("No Window !");
}
/************************************************************
*                     Open_Screen_and_Window()        (User)*
* Function: Editor Screen and Window close                  *
************************************************************/
VOID Close_Screen_and_Window()
{
   CloseWindow (Window);
   CloseScreen (Screen);
   CloseLibrary (IntuitionBase);
}
/************************************************************
*                          main()                     (User)*
*-----------------------------------------------------------*
* Input - Parameter:                                        *
* argv[1]: Disk drive (df0:, df1: etc.)                     *
* argv[2]: Output_file                                      *
************************************************************/
main (argc,argv)
UWORD argc;
BYTE      *argv[];
{
   UWORD i;
   ULONG Unit=0l;
   BYTE  *InputString;
   UWORD Len;
   InputString = argv[1];
   if (argc >= 2)
   {
      Len = strlen(InputString);
      for (i=0;i<Len;i++)
      {
         if ((*InputString >= 'A') && (*InputString <= 'Z'))
            *InputString |= (BYTE) 0x20; /* lowercase letters */
         InputString++;
      }
      if (strcmp ("df0:",argv[1]) == 0l) Unit = 0;
      else
      if (strcmp ("df1:",argv[1]) == 0l) Unit = 1;
      else
      if (strcmp ("df2:",argv[1]) == 0l) Unit = 2;
      else
      if (strcmp ("df3:",argv[1]) == 0l) Unit = 3;
   }
```

```
   if (argc == 3)
   {
      File = Open (argv[2],MODE_NEWFILE);
      if (File == 0l)
      {
         printf ("Can't open %s!",argv[2]);
         CloseIt (" ");
      }
   }
   if (argc > 3)
   {
      printf ("USAGE: %s [[DF?:] [ListFile]]!\n",argv[0]);
      exit(0);
   }
   Open_Screen_and_Window();
   SectorBuffer = (BYTE*) AllocMem ((ULONG)(TD_SECTOR),MEMTYPE);
      /* ffor one sector (Read/Write) */
   if (SectorBuffer == 0) CloseIt ("No SectorBuffer !");
   TrackBuffer  = (BYTE*) AllocMem (RAW_TRACK_LEN,MEMTYPE);
      /* for one track (RAWREAD/WRITE) */
   if (TrackBuffer == 0) CloseIt ("No TrackBuffer !");
   LabelBuffer  = (BYTE*) AllocMem ((ULONG)(NUMSECS),MEMTYPE);
      /* for Label-Area */
   if (LabelBuffer == 0) CloseIt ("No LabelBuffer !");
   for (i=0  ;i<32 ;i++) ConversionTable[i] = (BYTE)'.';
   for (     ;i<128;i++) ConversionTable[i] = (BYTE)i;
   for (     ;i<160;i++) ConversionTable[i] = (BYTE)',';
   for (     ;i<256;i++) ConversionTable[i] = (BYTE)i;
   ConsoleRead  = (struct IOStdReq *)GetDeviceBlock (CON_LEN);
   ConsoleWrite = (struct IOStdReq *)GetDeviceBlock (CON_LEN);
   ConsoleRead->io_Data    = (APTR) Window;
   ConsoleRead->io_Length  = (ULONG) (sizeof (struct Window));
   Open_A_Device ("console.device",0l,&ConsoleRead,0l,0l);
   Console_Copy (ConsoleRead,ConsoleWrite);
   Console_Write (ConsoleWrite,
      "Welcome to the wonderful World of TrackDisk !\n",-1l);
   The_Trackdisk_Device(Unit);
   Close_A_Device (ConsoleRead);
   FreeDeviceBlock (ConsoleWrite);
   if (File != 0l) Close (File);
   FreeMem (SectorBuffer,(ULONG)(TD_SECTOR));
   FreeMem (TrackBuffer ,RAW_TRACK_LEN);
   FreeMem (LabelBuffer ,(ULONG)(NUMSECS));
   Close_Screen_and_Window();
}
```

The program is called using its name and an argument representing a filename to which you would like the data saved. For example, the following command sequence invokes the disk editor, reads the disk in drive DF1: and creates a file named `listfile`:

```
DiskEd df1: listfile
```

The disk sector contents appear on the screen. Entering <d><Return> , writes the data to the file `listfile`. Pressing <h><Return> lists the command overview. The following command calls the disk editor and loads the sector data from the disk in DF0: (no file is created):

```
DiskEd
```

5. Standard File Formats

Software developers have invented file formats based on specific standards. File standards allow a user to pass data between drawing programs, word processors and even sound programs.

5.1 IFF

Most commercial software developers used their own file formats. It was easier to implement file systems in house, rather than try conforming to a standard. On one hand, this costs money because the buyer must finance each development. On the other hand, independent file formats make it impossible to exchange data between two different programs if they are incompatible. A universal file format would make applications more marketable for their flexibility.

Enter Electronic Arts

The staff of Electronics Arts discussed this very problem. The Apple Macintosh had a generally accepted set of formats for exchanging data between programs. EA felt that a file standard could be created for the Amiga that could be used for text, graphics and sound files alike.

The format is called IFF (Interchange File Format). Electronic Arts developed the format in 1984 and made it available to the public in January 1985. Developers have expanded IFF to fit the requirements of their programs. And although IFF saves more data in a file than other file formats under certain conditions, error free file reading is almost guaranteed.

This chapter splits IFF into many different forms. All are IFF, but all have their own qualities. We divided the chapter to keep parameters and IFF factors consistent with each format category.

The philosophy of IFF

All IFF systems have a standardized design. This makes processing simple. You'll always find the following items in an IFF file:

Header A header always exists at the beginning of each file. The header retains all the information that informs the program of data found in this file, as well as the type of data in the file (i.e., graphic data, sound data or text data).

Chunks Chunks comprise the remainder of the file. Chunks are blocks of data that contain groups of data. For example, an IFF file containing music data has a chunk outlining the parameters of each musical voice's sound capabilities. A graphic IFF file has a color chunk in which it stores all the color data needed for the graphic. Chunks provide the developer with a system of data building blocks which, like a set of toy building blocks, can be expanded as much as possible. This open format ensures that IFF will be around for a long time, since it is so open to expansion.

Each of the chunks discussed here is identified in the header by four characters. The data follows the chunk (we'll spend most of this chapter looking at chunk data).

5.1.1 The IFF ILBM graphic format

IFF's popularity is mostly attributable to the drawing program DeluxePaint®, which many users consider the standard among drawing programs. This is partly why many programs which make use of graphics use IFF for file management. Let's take a closer look at the ILBM (InterLeaved BitMap) and how it is constructed.

The form makes up a large part of an IFF graphic file. This form combines many chunks into a single file. And because all of the chunks belong to one graphic file, they are packed. The form contains a single item of information— the length of the data file and the data type. So the read routine can later determine if all of the data is actually present.

```
#define ID_ILBM MakeID('I','L','B','M')
```

Next comes the labeling of the ILBM format. The four letters "ILBM" indicate this particular format. The individual chunks of our data file follow. Let's look at them one at a time.

BMHD (BitMapHeaDer)

```
#define ID_BMHD MakeID('B','M','H','D')
```

This chunk contains the data that cover the formal attributes of our graphic. It is handled as a structure that is important later for opening the screen, because it acts as a storage space for measurements, depth and other values. Take a closer look at this structure:

```
typedef struct
   {
   UWORD w, h;                  /* Width and height       */
   WORD x, y;                   /* Position               */
   UBYTE nPlanes;               /* Depth of the screen    */
   Masking masking;             /* Contains mask flags    */
   Compression compression;     /* Compression Type       */
   UBYTE pad1;                  /* Fill byte              */
   UWORD transparentColor;      /* Number of trns. color */
   UBYTE xAspect, yAspect;      /* Aspects                */
   WORD pageWidth, pageHeight; /* Screen width & height */
   }
   BitMapHeader;
```

The `w` (width) and `h` (height) variables define the graphic's size in pixels. A graphic doesn't have to be saved at the same size as the screen. The brushes that are managed by the drawing program are also saved as ILBMs, and in most cases do not encompass the full screen dimensions.

The `x` and `y` variables specify the position of the graphic section. When an entire screen is saved, these parameters contain the value 0.

The `nPlanes` variable state the number of bit-planes used in the graphic. This is different from the colormap, because the number of colors calculated is derived from the number of bit-planes.

The `Masking` variable specifies the type of masking used by the graphic. You can select from `mskNone` (where no masking results) or `mskHasMask`. In addition to the normal bit-planes, a `MaskPlane` is saved that designates the masking for the graphic. The flag `mskHasTransparentColor` announces that the graphic includes a transparent color (the number of this transparent color lies in the `transparentColor` variable). A setting named `mskLasso` is taken from the Apple Macintosh. This encircles the graphic like a lasso. This makes it possible to remove the border of the graphic, which reduces the size of the graphic and saves memory.

```
typedef UBYTE Masking;
#define mskNone 0L
#define mskHasMask 1L
#define mskHasTransparentColor 2L
#define mskLasso 3L
```

The `compression` variable specifies the type containing the bit-map data. Either this variable contains nothing, which means the data is taken from the memory and saved exactly as it is, or a number which designates the method by which the data is keyed and packed.

```
typedef UBYTE Compression;
#define cmpNone 0L
#define cmpByteRun1 1L
```

The `Pad1` variable contains a fill byte, ensuring that the structure contains an even number of bytes. This byte is currently unused, so it contains zero. In every case this should be watched because later versions may make other use of this, and a value other than zero may cause problems.

The `xAspect` and `yAspect` variables contain the ratio between the X side and Y side of the graphic. This information is important for programs that transport graphics from one resolution to another, or transport graphics from one brand of computer to another brand altogether.

```
#define x320x200Aspect 10L
#define y320x200Aspect 11L
#define x320x400Aspect 20L
#define y320x400Aspect 11L
#define x640x200Aspect 5L
#define y640x200Aspect 11L
#define x640x400Aspect 10L
#define y640x400Aspect 11L
```

The `pageWidth` and `pageHeight` variables supply additional information about the graphic that can actually be larger or smaller than the screen on which it is displayed.

CMAP (ColorMAP)

```
#define ID_CMAP MakeID('C', 'M', 'A', 'P')
```

The colormap is the opposite of the bit-map header of a chunk that doesn't always have the same length. The length depends on how many bit-planes the graphic has, since the colormap computes the number of colors saved from the bit-planes.

Each color register stores three byte values (red, green and blue). Values for these section colors can range from 0 to 255. The Amiga doesn't have that many shades. But because the IFF format was developed for more than one computer, some extra flexibility was added. This is why we must move all of the color values into the higher placed bit (i.e., multiply it by 16). Each color register also has a structure:

```
typedef struct
   {
   UBYTE red, green, blue;
   }
   ColorRegister;
```

Please bear in mind that creating and reading an IFF file may give the colormap an odd number of bytes. Then the list must be completed with a null byte.

CRNG (ColorRaNG)

```
#define ID_CRNG MakeID('C','R','N','G')
```

It's possible to cycle through a color region, creating very interesting effects. The CRNG chunk supports this function. It gives an area in the color table that should be washed out. The structure for this looks like the following:

```
typedef struct
   {
   WORD pad1;
   WORD rate;
   WORD active;
   UBYTE low, high;
   } CRange
```

`pad1` supplies the necessary fill character.

`Rate` specifies the speed at which the colors are exchanged. For example, 16384 sets 60 changes per second. One rule: the larger the number, the more steps per second.

CCRT (Color Cycling Range and Timing)

```
#define ID_CRNG MakeID('C','C','R','T')
```

The CCRT chunk also controls color cycling. Both chunks are treated by independent tasks, depending on the manufacturer. Commodore-Amiga's GraphiCraft uses the CCRT chunk, while DeluxePaint uses the CRNG chunk.

We must read both chunks into our program and process them to make them completely compatible with colorcycling. Because both chunks are somewhat different, there is another structure:

```
typedef struct {
   {
   WORD direction;
   UBYTE start, end;
   LONG seconds;
   LONG microseconds;
   WORT pad;
   } CycleInfo;
```

The `direction` variable gives the direction in which the color should be cycled. 0 = no movement, 1 = forward and -1 = reverse. `Start` and `end` give the starting and ending number of both color registers between which color change occurs.

The CCRT chunk handles time differently from the CRNG chunk. Commodore-Amiga gives the `seconds` and `microseconds` variables. This is the same as the other functions in the Amiga library. Then this division also takes place there (look at the Preferences structure and the Intuition library).

`Pad` acts as the fill byte to keep the structure set at an even number of bytes.

Before we examine the most important of all of the chunks (the BODY chunk), we must address some lesser used chunks.

GRAB (GRAB position)

```
#define ID_GRAB MakeID('G','R','A','B')
```

The GRAB chunk indicates the relative position of the cursor. This is useful for placing brushes, which can be placed at any point on the screen.

```
typedef struct
   {
   WORD x, y;
   }
   Point2D;
```

The `x` and `y` coordinates specify the upper left corner of the graphic. The entire GRAB chunk consists only of this `point2D` structure.

DEST (DESTination bitplanes)

```
#define ID_DEST MakeID('D','E','S','T')
```

This chunk allows placement of the available bit-planes of the BODY chunk in other bit-planes of the graphic, while filling the unused bit-planes with 0 or 1 bit values. So you can easily change the colors of the original graphic. This chunk also consists of a data structure containing all of the applications:

```
typedef struct
   {
   UBYTE depth;
   UBYTE pad1;
   UWORD planePick;
   UWORD planeOnOff;
   UWORD planeMask;
   }
   DestMerge;
```

The `depth` variable designates the depth of the screen in which the data should be entered.

The `pad1` variable represents a fill byte (currently unused here).

The `planePick` variable examines the set bits. Each set bit is named: packs the next bit-plane from the file in the bit-plane of the screen with the number of the set bit. When a bit is not set, the same bit is considered under `planeOnOff`. When this is set the entire bit-plane is filled with 1, otherwise you clear it.

`planeMask` has the task of suppressing writing to a bit-plane. The corresponding bit for a bit-plane is set if it should be written in this variable. In the opposite case, this bit is cleared and the bit-plane remains undisturbed. The `planePick` and `planeMask` variables contain the default value `2^nPlanes - 1`. This ensures all set bits for each plane, as well as the use of all available bit-planes.

SPRT (SPRiTe)

```
#define ID_SPRT MakeID('S','P','R','T')
```

Sprites can also be saved in ILBM files with the help of this chunk. The chunk's single value designates the precedence of the sprite. The value 0 represents highest precedence. The higher the value, the lower the sprite precedence. The sprite with the highest precedence is graphically placed ahead of all others.

```
typedef UWORD SpritePrecedence;
```

CAMG (Commodore AMiGa computer)

```
#define ID_CAMG MakeID('C','A','M','G')
```

Unlike the other computers that have IFF file systems, the Amiga includes a set of `ViewModes`. These display modes include HAM and interlace mode. A new chunk (the CAMG chunk) compensates for the support not given by normal IFF ILBM chunks. The CAMG chunk contains only the `ViewMode` register:

```
typedef struct
   {
   ULONG ViewModes;
   }
   CamgChunk;
```

BODY (all Bit-planes and the Optional mask, interleaveD bY row)

```
#define ID_BODY MakeID('B','O','D','Y')
```

The BODY chunk contains the bit-map—the most important section of the IFF graphic file. This chunk operates under certain rules set by the other data.

After the number of the colors and bit-planes, the BODY chunk lists the data report. This report contains the bit-map in a linear (line oriented) format. This means that the report saves the first row of the first bit-plane, then the first row of the second bit-plane, and so on. When the first rows of all the bit-planes and the optional masks are saved, the next row is written in the same order (first bit-plane, second bit-plane, etc.). This continues until the last row of the last bit-plane. Then the BODY chunk ends.

This knowledge of the memory layout is very important, especially for graphics that consist of very large color areas. When you add the information about compression covered earlier in this chapter, you can actually save some memory when saving these files.

The following program reads and displays IFF files.

```
/***************************************************************/
/*              Simpler IFF-Read und Display                   */
/*                      (ONE SPEED-UP)                         */
/* (c) Bruno Jennrich                                          */
/***************************************************************/
#include "exec/types.h"
#include "exec/memory.h"
#include "exec/devices.h"
#include "devices/keymap.h"
#include "graphics/gfxmacros.h"
#include "graphics/regions.h"
#include "graphics/copper.h"
#include "intuition/intuition.h"
#include "graphics/gfxbase.h"
#include "graphics/gels.h"
#include "hardware/custom.h"
#include "hardware/blit.h"

struct IntuitionBase *IntuitionBase;
struct GfxBase       *GfxBase;

long                 *DosBase;
char Mask[9] = {0,1,2,4,8,16,32,32};           /* color number */
```

```
typedef struct BitMapHeader {
                                UWORD    w,h;
                                WORD     x,y;
                                UBYTE    BitPlanes;
                                UBYTE    Masking;
                                UBYTE    Compression;
                                UBYTE    PadByte;
                                UWORD    TransCol;
                                UBYTE    XAspect,YAspect;
                                WORD     Width,Height;
                              };
typedef struct ColorRegister {
                                UBYTE red;
                                UBYTE green;
                                UBYTE blue;
                               };
typedef struct CommodoreAmiga {
                                   UWORD PadWord;
                                   UWORD ViewModes;
                                };
struct BitMapHeader BMHD;
struct ColorRegister Colors[32];
struct CommodoreAmiga CAMG;
struct Screen *Screen;
struct NewScreen NewScreen;
LONG FileHandle;
LONG Len;
ULONG ChunkLen;
BOOL BMHDFlag;
BOOL CMAPFlag;
BOOL CAMGFlag;
BOOL BODYFlag;
BOOL FoundChunk;
BOOL ShowFlag = FALSE;
UBYTE Buffer [300];
LONG i;                 /* General Counter */
LONG x;                 /* Column Counter */
LONG y;                 /* Line Counter */
LONG b;                 /* BitPlane Counter */
UBYTE ByteCount;
UBYTE BytesPerRow;
char *WhereIsIt;        /* BitPlane Address */
char *MouseButton = (char *) 0xBFE001;
CloseIt (s)
char *s;
{
   printf ("%s\n",s);
   if (FileHandle != 0) Close (FileHandle);
   if (Screen != 0) CloseScreen (Screen);
   if (DosBase != 0) CloseLibrary (DosBase);
   if (IntuitionBase != 0) CloseLibrary (IntuitionBase);
   if (GfxBase != 0) CloseLibrary (GfxBase);
   exit (0);
}
Lread (Buffer,Num,Flag)
LONG Buffer;
WORD Num;
BOOL Flag;
{
   Len = Read (FileHandle,Buffer,Num);
   if ((Flag == TRUE) && (Len < 0)) CloseIt ("File-Error
!!!!!!!\n");
```

```
}


main (argc, argv)

int argc;                    /* Argument Counter */
char **argv;                 /* Argument Value */

{
   if (argc != 2)
      {
         printf (" USAGE: \"ShowILBM IFF-Filename\"\n");
      }
   else
      {
   printf (" End by clicking in the upper left hand corner.\n");
    DosBase= (LONG *)  OpenLibrary("dos.library",0);
    GfxBase=(struct GfxBase *)OpenLibrary("graphics.library",0);
    IntuitionBase=(struct IntuitionBase *)
OpenLibrary("intuition.library",0);
   if ((DosBase == 0) || (GfxBase == 0) || (IntuitionBase == 0))
            CloseIt ("Librarys ?!?!?!?!?!?!?!?!?!?!?\n");
         FileHandle = Open (argv[1],1005);

         if (FileHandle == 0) CloseIt ("File OPEN Error !\n");
         Lread (Buffer,12,TRUE);

         if (strncmp(&Buffer[0],"FORM",4) != 0) CloseIt ("Not
IFF-File !!!\n");
         if (strncmp(&Buffer[8],"ILBM",4) != 0) CloseIt ("Not
ILBM-File !!!\n");
         BMHDFlag = FALSE;
         CMAPFlag = FALSE;
         CAMGFlag = FALSE;
         BODYFlag = FALSE;
       Loop:
         FoundChunk = FALSE;
         Lread (Buffer,8,FALSE);
         if (Len <= 0)
            if ((BMHDFlag == TRUE) && (BODYFlag == TRUE) &&
(CMAPFlag == TRUE))
              {
              if (CAMGFlag == FALSE) printf (" No CAMG !!!\n");

               LoopA:
                 while ((*MouseButton & 0x40) == 0x40);
             if ((Screen->MouseX == 0) && (Screen->MouseY == 0))
                    CloseIt ("");
                 else goto LoopA;
              }
            else
              {
                if ((BMHDFlag != TRUE) || (BODYFlag != TRUE) ||
                     (CMAPFlag != TRUE))
                       {
             if (BMHDFlag == FALSE) CloseIt (" No BMHD !!!\n");
             if (BODYFlag == FALSE) CloseIt (" No BODY !!!\n");
             if (CMAPFlag == FALSE) CloseIt (" No CMAP !!!\n");
                       }
              }
            ChunkLen =
Buffer[4]*16777216+Buffer[5]*65536+Buffer[6]*256+Buffer[7];
```

```
                if (strncmp (Buffer,"BMHD",4) == 0)
                   {
                  if (BMHDFlag == TRUE) CloseIt (" Two BMHD's ?\n");
                      Lread (&BMHD,ChunkLen,TRUE);    /* BMHD Read */
                      NewScreen.LeftEdge = 0;
                      NewScreen.TopEdge = 0;
                      NewScreen.Width = BMHD.Width;
                      NewScreen.Height = BMHD.Height;
                      NewScreen.Depth = BMHD.BitPlanes;
                      if (CAMGFlag == TRUE) NewScreen.ViewModes =
CAMG.ViewModes;
                      else
                         {
                            NewScreen.ViewModes = 0;
                            if (NewScreen.Width > 320)
                               NewScreen.ViewModes |= HIRES;
                            if (NewScreen.Height > 200)
                               NewScreen.ViewModes |= LACE;
                         }
                      NewScreen.Type = CUSTOMSCREEN;
                      NewScreen.Font = NULL;
                      NewScreen.DefaultTitle = (UBYTE *) argv[1];
                      NewScreen.Gadgets = NULL;
                      NewScreen.CustomBitMap = NULL;
                  Screen = (struct Screen*) OpenScreen (&NewScreen);
                      if (Screen == 0) CloseIt (" No Screen !!!\n");
                      BytesPerRow = BMHD.Width/8;
                      ShowTitle (Screen,FALSE);
                      BMHDFlag = TRUE;
                      FoundChunk = TRUE;
                   }
                if (strncmp (Buffer,"CMAP",4) == 0)
                   {
                    if (CMAPFlag == TRUE) CloseIt("Two CMAP's ?\n");
                      if (BMHDFlag == FALSE) CloseIt("BMHD must be
before CMAP  !!!\n");
                      Lread (Colors,ChunkLen,TRUE);   /* BMHD read */

                      for (i=0; i<Mask[BMHD.BitPlanes]; i++)
SetRGB4(&Screen->ViewPort,i,Colors[i].red>>4,Colors[i].green>>4,
Colors[i].blue>>4);
                      CMAPFlag = TRUE;
                      FoundChunk = TRUE;
                   }
                if (strncmp (Buffer,"BODY",4) == 0)
                   {
                 if (BODYFlag == TRUE) CloseIt ("Two Body's ???\n");
                      if (BMHDFlag == FALSE)
                      CloseIt ("BMHD must come before BODY  !!!\n");
                      for (y=0;y<BMHD.Height;y++)
                            for (b=0;b<BMHD.BitPlanes;b++)
                               {
                                  ByteCount = 0;
                                  WhereIsIt =
                                  (char *) Screen->RastPort.BitMap-
>Planes[b]+y*BytesPerRow;
                                  if (BMHD.Compression == 0)
                                     {
                                Lread (WhereIsIt,BytesPerRow,TRUE);
                                     }
                                  if (BMHD.Compression == 1)
```

```
                        while (ByteCount<BytesPerRow)
                          {
                           Lread (&Buffer[0],1,TRUE);
                             if (Buffer[0] < 128)
                                {
        Lread (WhereIsIt+ByteCount,Buffer[0]+1,TRUE);
              ByteCount += Buffer[0]+1;
                                }
                 /* Buffer[0] == 128 => Nop */
                             if (Buffer[0] > 128)
                                {
                Lread (&Buffer[1],1,TRUE);
     for (i=ByteCount;i<(ByteCount+257-Buffer[0]);i++)
            *(WhereIsIt+i) = Buffer[1];
             ByteCount += 257-Buffer[0];
                             }
                        }
                  }
        BODYFlag = TRUE;
        FoundChunk = TRUE;
     }
   if (strncmp (Buffer,"CAMG",4) == 0)
     {
    if (CAMGFlag == TRUE) CloseIt ("Two CAMG's !!!\n");
        if (BMHDFlag == FALSE)
         CloseIt ("BMHD must come before CAMG !!!\n");
        Lread (&CAMG,ChunkLen,TRUE);
        Screen->ViewPort.Modes = CAMG.ViewModes;
        RemakeDisplay();
        CAMGFlag = TRUE;
        FoundChunk = TRUE;
     }
   if (FoundChunk == FALSE)
     {
        Lread (Buffer,ChunkLen,FALSE);
        if ((ChunkLen & 1) == 1) Lread
(Buffer,1,FALSE);
     }
   goto Loop;
  }
}
```

Program description

The program uses two universally supported routines. The `CloseIt()` function closes the program in case of an error or the end of the program. It checks what was opened and closes open files to prevent system errors. The `ReadIt()` function reads a certain quantity of data.

The main program begins by opening the libraries and files, then reading the data. First a test is performed for the existence of an IFF ILBM , then the first chunk in the main loop is selected. If recognized by the program, its data is read in according to the rules described above. Unknown chunks or those not supported by the program are simply ignored.

When the necessary data is present, the BMHD reader opens a screen in which the bit-map data with the BODY chunk is entered. The program waits for a mouse click in the upper left corner to close everything and create its work. Next we see the ILBM format of an arrangement of all of the chunks that can be encountered when you read an ILBM FORM. Remember when writing that all of the chunks that are read, including those that your program does not process, are written back to disk.

```
#ifndef ILBM_H
#define ILBM_H
#define ID_ANFR MakeID('A','N','F','R')
#define ID_MAHD MakeID('M','A','H','D')
#define ID_MFHD MakeID('M','F','H','D')
#define ID_CM16 MakeID('C','M','1','6')
#define ID_ILBM MakeID('I','L','B','M') /* Interleaved BitMap */
#define ID_ILBM MakeID('S','H','A','K') /* Shakespeare-Chunk,
                                           that contain the
                                           ILBMs */
#define ID_ANIM MakeID('A','N','I','M') /* Animation format */
#define ID_BMHD MakeID('B','M','H','D') /* BitMap Header */
#define ID_ANHD MakeID('A','N','H','D') /* Animations Header */
#define ID_CMAP MakeID('C','M','A','P') /* Color Map */
#define ID_GRAB MakeID('G','R','A','B') /* Hot Spot of the
                                           BitMap */
#define ID_DEST MakeID('D','E','S','T') /* Bitplane
                                           distribution */
#define ID_SPRT MakeID('S','P','R','T') /* Sprite recognition */
#define ID_CAMG MakeID('C','A','M','G') /* Commodore Amiga
                                           View */
#define ID_BODY MakeID('B','O','D','Y') /* BitMap Data */
#define ID_ATXT MakeID('A','T','X','T') /* are used */
#define ID_PTXT MakeID('P','T','X','T') /*    by time */
#define ID_DLTA MakeID('D','L','T','A') /* Anim Delta
                                           movement */
#define ID_CRNG MakeID('C','R','N','G') /* Color Cycling
                                            Chunk */
```

We'll conclude this part by mentioning a few problems with IFF, and possible solutions.

1.) Display screen size

Usually the size of the graphic and the size of the screen are found in the bit-map header, on which the graphic is constructed. The value in `w` and `h` always correspond to the number of pixels in the x and y directions of the graphic. The values in `pageWidth` and `pageHeight` comprise the number of pixels of the screen in the x and y directions. The values from DPaintII are also written in the structure. This makes sense, because you can save a picture that is much larger than the screen and display on a 320 x 200 pixel screen using them.

Many programs that also support the `OverScan` mode, enter larger values in the `pageWidth` and `pageHeight` than can be selected because of the `OverScan`. This causes some problems because although the value is over 320, no HIRES screen has to be opened. To get around this problem we recommend that you save the `ViewMode` and load it when reloading the graphic. This keeps the selected resolution clearly defined. Make sure your program reads the CAMG chunk as well.

2.) Forgotten chunks

Many non-Amiga IFF compatible applications simply ignore the CAMG chunk, although the Amiga may not use the same resolution as the other programs. The HAM or HALFWIDTH graphics are saved by section. The developer assumes that the program recognizes six bit-planes. No thought was given to the other modes, because the program should be able to distinguish a HAM graphic from a HALFWIDTH graphic.

Another problem is that many programs write directly in the Screen Register when they are saved. This can cause problems when the flags `SPRITES`, `VP_HIDE`, `GENLOCK_AUDIO` and `GENLOCK_VIDEO` are set. You clear these flags when you save the CAMG chunk.

3.) Color cycling

DeluxePaintII writes all of the cycling ranges on the diskette as active without giving any thought as to if the picture is cycled or not. This saves the CRNG chunk incorrectly.

That makes it impossible for other programmers to determine whether a picture should be actively cycled or not, because all of the graphics created in DeluxePaintII suffer from this problem. The only solution is through an extra setting with the Slide Show program, either as a parameter in the CLI or setting a ToolType with the entry CYCLING=ON.

4.) The number of colors of a CMAP chunk

Other colormap problems occur when using HAM pictures. This should contain all of the colors needed for the picture. But although six bit-planes are used, this graphic type only uses 16 colors. There is some disagreement about this because many programs save 16 colors, some save 32 and still others save 64, because an IFF rule calculates the number of colors based on 1<<bitmap depth, and that results in 64.

A CMAP chunk never has the complete number of colors that can be given through the BMHD chunk. Read the byte statement at the beginning of the chunk, and never place the number of the color register after the entry in the CMAP.

5.1.2 The IFF FTXT text format

This IFF format was developed to allow free file exchange between word processing systems. Unfortunately this format didn't catch on. For example, WordPerfect® and BeckerText file types are incompatible. The only program that uses the FTXT format is TextCraft, which was initially bundled with the Amiga 1000.

The FTXT format has the following format:

```
FORM####
FTXT
[FONS]####FontData
CHRS####Characters....<END>
(#### equals file length or chunk length)
```

The `FTXT` string indicates that this file is a formatted text file (FTXT). The two chunks supported by the FTXT format are `CHRS` and `FONS`.

The `FONS` chunk consists of a font specifier structure that determines which font should be used:

```
Offset    Structure
------    --------
          FontSpecifier
          {
0 0x00      UBYTE id;          /* Font number /* 0 - 9 */
1 0x01      UBYTE pad1;
2 0x02      UBYTE proportional; /* Proportional font ? */
                       /* 0 = unknown, 1 = yes, 2 = no */
3 0x03      UBYTE serif;        /* Serifs ? */
                       /* 0 = unknown, 1 = yes, 2 = no */
4 0x04    char name[];   /* Font name (e.g. "topaz/8") */
...       }
```

The length of this chunk depends on the number of characters in the font name. The brackets surrounding the `FONS` keyword indicate that this chunk is optional in creating a complete IFF file.

The `CHRS` chunk consists of the actual text (ASCII codes 0x20 to 0x7f). The number of characters contained in this chunk follow the `CHRS` mark. `CHRS` and `FONS` chunks can be swapped in an FTXT file.

5.1.3 The IFF SMUS music format

This format allows the exchange of musical compositions between music development applications. The system must determine the notation of each voice, as well as the instrumental quality of each voice.

An SMUS file has the following format:

```
FORM####
SMUS
SHDR####SScoreHeader
[NAME]####".."
[(c) ]####".."
[AUTH]####".."
[IRev]####????
ANNO####".."
INS1####RevInstrument
TRAK####SEvents...<End>
```

The chunks

SMUS indicates that this IFF file is an SMUS (Simple MUsic Score) file.

The SHDR chunk contains a ScoreHeader structure:

```
Offset    Structure
------    --------
          struct SScoreHeader
          {
0  0x00      UWORD tempo;
2  0x02      UBYTE volume;  /* volume (0-127) */
3  0x03      UBYTE ctTrack; /* number of voices */
4  0x04   }
```

The tempo variable of a piece of music is given in increments of a quarter note note per 128 minutes. If tempo = 1, a single quarter note plays for a 128-minute period.

The NAME chunk contains the name of the piece of music (e.g., "Fugue in D").

The (c) chunk contains the copyright notice (e.g., "(c) Helmut Beethoven 1988").

The AUTH chunk contains the name of the author/composer (e.g., "Michael Sting").

The IRev chunk can be used for storing other information pertaining to the composition (e.g., revision).

The ANNO chunk contains comments (annotations) about the piece of music. This chunk must be present, but can consist of 0 bytes.

The INS1 chunk contains data about the instrument to be used. The following structure specifies the instrument data:

```
Offset     Structure
------     --------
           struct RefInstrument
           {
0  0x00       UBYTE register;
1  0x01       UBYTE type;
2  0x02       UBYTE data1,
3  0x03             data2;
4  0x04       char name[];
           }
```

The register variable contains the number of the instrument. This allows the selection of a new instrument as a voice is playing.

The type variable lets you specify the instrument name (type = 0) or the use of a MIDI channel (type = 1). In the latter case, the bytes data1 and data2 designate the MIDI channel and the MIDI preset to be used. The name array is not used in conjunction with MIDI.

The INS1 sets the instrument type, but not the tuning. Instrument 0 represents voice 0, instrument 1 represents voice 1, etc. This order can be changed later.

The TRAK chunk contains the notes to be played and other information. Each entry in TRAK is two bytes long. The first byte specifies how the second byte should be interpreted:

```
Offset     Structure
------     --------
           struct SEvent /* Simple Musical Event */
           {
0 0x00        UBYTE SID;
1 0x01        UBYTE data;
2 0x02     }
```

Here are the values allowed within SID's SEvents:

0-127 (note) These values specify the pitch of the tone to be played. The data byte designates the length of the tone:

Bit 7 If this byte is set, the current note and the following note are played as a chord.

Bit 6 If this bit is set, the current note and the following note are played without interruption.

Bits 4 and 5

These bits test for unusual note rhythms:

Note value	Binary	Decimal
Triplet	01	1
Quintuplet	10	2
Septuplet	11	3
"Normal" note	00	0

Bit 3 If this bit is set, the current note is played as a dotted note (a dot makes the note 1.5 times its normal duration).

Bits 0—2

These bits indicate note length:

Note value	Binary	Decimal
Whole	000	0
Half	001	1
Quarter	010	2
Eighth	011	3
Sixteenth	100	4
Thirty-second	101	5
Sixty-fourth	110	6
128th	111	7

128 (rest) If `SID` contains the value 128, a rest is played. The duration of the rest length is specified in the `data` byte.

129 (instrument)

This `SEvent` changes the instrument for this particular voice. The `data` byte contains the number of the instrument.

130 (time) This `SEvent` states the time. The quotient of the top 5 and bottom 3 bits give the time. The top 5 bits are given in beats per second (1-32), while the bottom 3 bits are given in `SID` = note. To create 4/4 time, the value 3 must be given in the top 5 bits and the value 2 (quarter note) in the bottom 3 bits. A value of 0 in the top 5 bits specifies one beat per second.

131 (pitch) This `SEvent` establishes the pitch of a note:

Number	Pitch
0	C
1	G
2	D
3	A
4	E
5	B
6	F#
7	C#
8	F
9	Bb
10	Eb
11	Ab
12	Db
13	Gb
14	Cb

132 (volume) This `SEvent` assigns a new volume to this voice. Values can range from 0 to 127.

133 (MIDI channel)
This `SEvent` allows you to select a new MIDI channel for subsequent notes (`data` = 0-255).

134 (MIDI presets)
This `SEvent` allows you to select new MIDI presets (`data` = 0-255).

5.1.4 The IFF 8SVX sample format

The 8SVX IFF format is used for the open exchange of digitized sounds between sampler/sound digitizing applications.

An 8SVX (8-bit Sampled VoX [Voice]) IFF file has the following format:

```
FORM####
8SVX
VHDR####Voice8Header
[NAME]####".."
[(c) ]####".."
[AUTH]####".."
ANNO####".."
ATAK####EGPoint
RLSE####EGPOINT
BODY####Samples...<End>
```

The `8SVX` chunk is the identification string for 8-bit sampled voice files.

The VHDR chunk contains a Voice8Header structure:

```
Offset    Structure
------    --------
          struct Voice8Header
          {
0  0x00      ULONG oneShotHiSamples,
4  0x04            repeatHiSamples,
8  0x00            samplesPerHiCycle;
12 0x0c   UWORD samplesPerSec;    /* Sampling Frequency */
14 0x0xe     UBYTE ctOctave,       /* number of octaves */
15 0x0f       sCompression;       /* Data compression ? */
16 0x10     LONG  volume;            /* see EGPoint.dest */
20 0x14   }
```

The NAME chunk contains the name of the sound (e.g., "AAAAHHHH!").

The (c) chunk contains the copyright notice (e.g., "(c) Dirty Harry").

The AUTH chunk contains the names of the author (e.g., "Jim Cottonfield III").

The ANNO chunk contains comments (annotations) about the sound. This chunk must be nominally present but can consist of 0 bytes.

The ATAK chunk contains the EGPoint structure which allows you to control the attack of the sound (start at a certain volume and reach a certain volume in a certain amount of time). The EGPoint structure looks like this:

```
Offset    Structure
------    --------
          struct EGPoint
          {
0  0x00      UWORD duration; /* in Milliseconds */
2  0x02      LONG  dest;     /(* volume factor */
6  0x06   }
```

The duration variable specifies the time the volume has to reach the new value. The dest variable specifies the factor by which the volume should be increased. This latter variable is a fixed-point variable: The top 16 bits indicate the amount to the left of the decimal point, while the bottom 16 bits indicate the amount to the right of the decimal point. A value of 0x00015000 means a factor of 1.5.

The RLSE chunk contains an EGPoint structure used for controlling the decay (volume fade).

The BODY chunk contains the sample itself.

6. The Amiga Libraries

This chapter describes all available library functions. The functions are arranged according to the libraries, and within the libraries they are arranged according to function groups. The basic layout is as follows:

Library: Description of the library's purpose.

Functions: Listing of all of the function divisions in the group, including the number of the page on which the function is printed.

Description: Descriptions of the functions according to group.

Structures: Listing of all of the important structures that are used by the functions of this library. The name of the include file which contains the definition follows the structure, enclosed in <>. A hexadecimal and decimal offset of the start of the structure elements in bytes precedes each structure element, so that you can find the elements with your debugger. Rember structures can change, so do not depend on addresses of system structures.

Each function is described as follows:

Name: Short description.

Syntax: Syntax of the function, the assembly language registers, the offset and the parameter types.

Description: Description of the function's exact purpose.

Parameter: Listing of the parameters needed by the function.

Result: Description of the value(s) returned by the function (if any).

Explanation: Description of any other data returned by the function (e.g., errors).

Warning: You must remember this information when you call the function. Read this if you read nothing else in the function description.

Comments: Additional commentary goes here, if any is needed.

Example: Demonstration program code (if needed).

See Also: Refers the reader to other functions in this book.

6.1 The exec library

The exec library is the only library that we can access without the `OpenLibrary()` command. The system has a pointer to your basis address at memory location 0x00000004. It is the only consistent address in the entire Amiga operating system.

When you program in C, you can use all of the commands. The compiler takes the basis address and correctly calls all of the functions. In assembly language you simply load the value from the named address and use this as the basis address.

This library performs all of the elementary system tasks. It supports other library tasks like data exchange and multitasking, and more. Its main purpose lies in executing the Amiga's essential survival tasks.

Exec Library Functions

6.1.1 Special functions

6.1.2 Interrupt functions

6.1.3 Memory functions

6.1.4 List management functions

6.1.5 Task functions

6.1.6 Message functions

6.1.7 Library functions

6.1.8 Device functions

6.1.9 Resource functions

6.1.10 Semaphore supported functions

6.1.11 Kickstart and memory functions

6.1.12 Additional functions

Before we go on to the functions, let's look at the structure which acts as this function's base:

```
struct ExecBase <exec/execbase.h>
{
0x00  00  struct Library LibNode;/* Library structure  */
0x22  34  UWORD SoftVer;        /* Version number of Kickstart */
0x24  36  WORD LowMemChkSum;    /* Checksum for the bottom
                                     range of memory */
0x26  38  ULONG ChkBase;        /* System basis address */
0x2A  42  APTR ColdCapture;     /* Cold start vector */
0x2E  46  APTR CoolCapture;
0x32  50  APTR WarmCapture;     /* Warm start vector */
0x36  54  APTR SysStkUpper;     /* System stack: top limit */
0x3A  58  APTR SysStkLower;     /* System stack: bottom limit */
0x3E  62  ULONG MaxLocMem;      /* Maximum accessible memory */
0x42  66  APTR DebugEntry;      /* Debugger's starting address */
0x46  70  APTR DebugData;       /* Debugger pointer and data */
0x4A  74  APTR AlertData;       /* Alert data pointer */
0x4E  78  APTR MaxExtMem;   /* Maximum allowable expansion RAM */
0x52  82  UWORD ChkSum;         /* Operating system checksum */
0x54  84  struct IntVector IntVects[16]; /* Interrupt vector
                                             table */
0x114  276 struct Task *ThisTask;/* Pointer to active task */
0x118  280 ULONG IdleCount;     /* Unused counter */
0x11C  284 ULONG DispCount;     /* Dispatch counter */
0x120  288 UWORD Quantum;       /* Processor time for each task*/
0x122  290 UWORD Elapsed;       /* Elapsed time units */
0x124  292 UWORD SysFlags;
0x126  294 BYTE IDNestCnt;
0x127  295 BYTE TDNestCnt;
0x128  296 UWORD AttnFlags;
0x12A  298 UWORD AttnResched;
0x12C  300 APTR ResModules;
0x130  304 APTR TaskTrapCode;   /* Task pointer */
0x134  308 APTR TaskExceptCode;
0x138  312 APTR TaskExitCode;
0x13C  316 ULONG TaskSigAlloc;  /* Key signal/traps */
0x140  320 UWORD TaskTrapAlloc;
0x142  322 struct List MemList; /* System lists */
0x146  336 struct List ResourceList;
0x15E  350 struct List DeviceList;
0x16C  364 struct List IntrList;
0x17A  378 struct List LibList;
0x188  392 struct List PortList;
0x196  406 struct List TaskReady;
0x1A4  420 struct List TaskWait;
0x1B2  434 struct SoftIntList SoftInts[5];
0x202  514 LONG LastAlert[4];          /* Last alert number */
0x212  530 UBYTE VBlankFrequency;      /* Video frequency */
0x213  531 UBYTE PowerSupplyFrequency; /* Power supply freq. */
0x214  532 struct List SemaphoreList;
0x222  546 APTR KickMemPtr;
0x226  550 APTR KickTagPtr;
```

```
0x22A  554 APTR KickCheckSum;          /* Kickstart checksum */
0x22C  558 UBYTE ExecBaseReserved[10];
0x238  568 UBYTE ExecBaseNewReserved[20];
0x24C  588
};
SYSBASESIZE ((long)sizeof(struct ExecBase))
Processor_Bitnumber:
AFB_68010 0L
AFB_68020 1L
AFB_68881 4L
Processor_Flags:
AFF_68010 (1L<<0)
AFF_68020 (1L<<1)
AFF_68881 (1L<<4)
Bytes:
AFB_RESERVED8 8L
AFB_RESERVED9 9L
```

6.1.1 Special functions

InitCode — **Initializes a resident code module**

Syntax:

```
InitCode(StartClass, Version);
   -72       D0          D1
ULONG StartClass;
ULONG Version;
```

Description: This function initializes all of the modules with the given features, and a version number greater than or equal to the one given.

Parameters:

StartClass: Flag value with class of code (warmstart, coldstart, coolstart).
Version: Version number.

InitStruct — **Initializes a table in memory**

Syntax:

```
InitStruct(InitTable, Memory, Size);
  -78          A1        A2     D0
ULONG *InitTable;
ULONG *Memory;
ULONG  Size;
```

Description: This function initializes a table at the memory location of a structure based on the given values.

Parameters:

InitTable: Pointer to the data table.
Memory: Pointer to the memory region.
Size: Size of the structure to initialize.

FindResident — Search resident module for its name

Syntax:

```
Resident = FindResident(Name);
    D0            -96     A1
ULONG *Resident;
char *Name;
```

Description: This function searches through the system list for a resident module with the given name.

Parameters: Name: Pointer to the name being searched for.

Result: Resident: Address of the found resident module, or null if none was found.

InitResident — Initializes a resident module

Syntax:

```
InitResident(Resident, SegList);
   -102         A1        D1
ULONG *Resident;
struct List *SegList;
```

Description: This function initializes a resident module.

Parameters:

Resident: Pointer to the module.
SegList: Pointer to a segment list.

```
struct Resident <exec/resident.h>
{
0x00  00  UWORD rt_MatchWord;
0x02  02  struct Resident *rt_MatchTag;
0x06  06  APTR rt_EndSkip;
0x0A  10  UBYTE rt_Flags;
0x0B  11  UBYTE rt_Version;
0x0C  12  UBYTE rt_Type;
0x0D  13  BYTE rt_Pri;
0x0E  14  char *rt_Name;
0x12  18  char *rt_IdString;
0x16  22  APTR rt_Init;
0x1A  26
};
RTC_MATCHWORD 0x4AFCL
RTF_AUTOINIT (1L<<7)
RTF_COLDSTART (1L<<0)
RTM_WHEN 3L
RTW_NEVER 0L
RTW_COLDSTART 1L
```

Alert — Informs user of an error

Syntax:

```
Alert(AlertNum, Parameters);
-108        D7       A5
ULONG AlertNum;
ULONG *Parameters;
```

Description: This function displays an error message to the user. For this it performs all necessary work.

Parameters:
AlertNum: Number describing the error.
Parameters: Pointer to the parameters.

Debug	**Starts system debugger**

Syntax:
```
Debug();
-114
```

Description: This function starts the system debugger. If the address was changed through `SetFunction()`, then another debugger can be used.

6.1.2 Interrupt functions

Disable	**Disables system interrupts**

Syntax:
```
Disable();
-120
```

Description: This function disables interrupts within the system.

Result: All interrupts are suppressed until either `Enable()` is called or the task moves into `Wait` status.

Comments: There should be one `Enable()` call for each call of `Disable()`.

See Also: `Enable(),Forbid(),Permit()`

Enable	**Enables system interrupts**

Syntax:
```
Enable();
-126
```

Description: This function enables interrupts within the system.

Result: All interrupts are accessible until `Disable()` is called.

Comments: There should be one `Disable()` call for each call of `Enable()`.

See Also: `Disable(),Forbid(),Permit()`

Forbid **Forbids task rescheduling**

Syntax:
```
Forbid();
-132
```

Description: This function forbids the dispatching program from assigning a certain amount of execution time to each task until `Permit()` is called.

Explanation: Other tasks also receive execution time if the called tasks are set in the `Wait` status.

See Also: `Permit(),Disable(),Enable()`

Permit **Permits task rescheduling**

Syntax:
```
Permit()
-138
```

Description: This function negates a `Forbid()` and allows the dispatcher to give other programs processor time. Only one call of `Permit()` may be used for each call of `Forbid()`.

See Also: `Forbid(),Disable(),Enable()`

SetSR **Sets/changes processor status register**

Syntax:
```
OldSR = SetSR(NewSR, Mask)
 D0      -144   D0    D1
LONG OldSR;
LONG NewSR;
LONG Mask;
```

Description: This function allows changes to the value of the CPU's status register.

Parameters:

NewSR: New bit values for status register.
Mask: Contains set bits where the value of `NewSR` should be transmitted.

Result: OldSR: The value of the status register before the call.

Example: You get the value of SR: actual = SetSR(0L, 0L);.

SuperState **Enables supervisor status**

Syntax:
```
OldSysStack = SuperState()
   D0            -150
```

Description: This function sets the processor to supervisor status.

Result: OldSysStack: Value restored when user status returns.

Explanation: The function does not work when you are already in the supervisor status. Then the value of OldSysStack is zero.

See Also: `UserState()`

UserState — Enables user status

Syntax:

```
UserState(SysStack)
-156      D0
```

Description: This function sets the processor to user status.

Parameter: SysStack: Value returned from `SuperState()`.

See Also: `SuperState()`

SetIntVector — Sets system interrupt vector

Syntax:

```
OldInterrupt = SetIntVector(IntNumber, Interrupt)
    D0           -162         D0          A1
struct Node *OldInterrupt;
LONG IntNumber;
struct Node *Interrupt;
```

Description: This function defines a new interrupt based on the `IntNumber` parameter.

Parameters:

IntNumber: Interrupt number (first five bits only).

Interrupt: Pointer to the interrupt node structure, containing the jump point for the interrupt handler and a pointer to the data segment.

Result:

OldInterrupt: Pointer to the old node structure which previously defined the interrupt.

```
struct Interrupt <exec/interrupts.h>
{
0x00  00  struct Node is_Node;
0x0E  14  APTR is_Data;  /* pointer to the data of the server */
0x12  18  VOID (*is_Code)();  /* Program code beginning of the
                                  server */
0x16  22
};
struct IntVector <exec/interrupts.h>
{
0x00  00  APTR iv_Data; /* not tested for the general use*/
0x04  04  VOID (*iv_Code)();
0x08  08  struct Node *iv_Node;
0x0C  12
};
struct SoftIntList <exec/interrupts.h>
{
0x00  00  struct List sh_List; /* Not tested for the general
                                   use*/
```

```
0x0E  14  UWORD sh_Pad;
0x10  16
};
SIH_PRIMASK (0xf0L)
INTB_NMI 15L
INTF_NMI (1L<<15)
```

AddInitServer — Inserts interrupt server

Syntax:

```
AddIntServer(IntNumber, Interrupt)
   -168          D0        A1
LONG IntNumber;
struct Node *Interrupt;
```

Description: This function adds another interrupt to the interrupt server list. The priority of the interrupt is tested when it is executed.

Parameters:

IntNumber: Interrupt number.
Interrupt: Pointer to the interrupt node structure.

See Also: `RemIntServer(),Cause(),SetIntHandler()`

RemIntServer — Removes interrupt server

Syntax:

```
RemIntServer(IntNumber, Interrupt)
   -174         D0         A1
LONG IntNumber;
struct Node *Interrupt;
```

Description: This function removes the given interrupt structure from the interrupt server list.

Parameters:

IntNumber: Interrupt number.
Interrupt: Pointer to the interrupt node structure.

See Also: `AddIntServer(),Cause(),SetIntHandler()`

Cause — Executes a software interrupt

Syntax:

```
Cause(Interrupt)
-180      A1
struct Node *Interrupt;
```

Description: This function causes a software interrupt to execute.

Parameters:

Interrupt: Pointer to an initialized interrupt node structure.

See Also: `AddIntServer(),RemIntServer(),SetIntHandler()`

6.1.3 Memory functions

Allocate — **Allocates memory block**

Syntax:

```
MemoryBlock = Allocate(FreeList, ByteSize);
     D0         -186      A0        D0
ULONG *MemoryBlock;
struct MemHeader *FreeList;
ULONG ByteSize;
```

Description: This function searches through the `FreeList` for a memory block of the desired size, allocates this block and returns a pointer.

Parameters:

FreeList: Pointer to a memory list header.
ByteSize: Size of the desired memory block in bytes.

Result:

MemoryBlock:
Pointer to the allocated memory block. This parameter contains zero if an error occurs.

See Also: `Deallocate()`, `AllocAbs()`, `AllocMem()`, `AllocEntry()`, `AllocRemember()`

```
struct MemHeader <exec/memory.h>
{
0x00  00  struct Node mh_Node; /* Node to add more  MemHeader */
0x0E  14  UWORD mh_Attributes; /* Type of this memory region */
0x10  16  struct MemChunk *mh_First;  /* first element of this
                                         memory list */
0x14  20  APTR mh_Lower;  /* top and bottom list of the entire
                             memory region */
0x18  24  APTR mh_Upper;
0x1C  28  ULONG mh_Free;  /* number of all the free bytes of
                             this memory region */
0x20  32
};
struct MemChunk <exec/memory.h>
{
0x00  00  struct MemChunk *mc_Next; /* left to the next memory
                                       chunk */
0x04  04  ULONG mc_Bytes;  /* size of the chunk in bytes */
0x08  08
};
```

Deallocate — **Releases memory block**

Syntax:

```
Deallocate(FreeList, MemoryBlock, ByteSize);
    -192       A0         A1          D0
struct MemHeader *FreeList;
ULONG *MemoryBlock;
ULONG ByteSize;
```

Description: This function releases the previously allocated memory block and inserts it in the `FreeList` again.

Parameters:
FreeList: Pointer to a memory free list.
MemoryBlock: Pointer to the memory block.
ByteSize: Size of the memory block.

See Also: `Allocate()`

AllocMem — Allocates system memory

Syntax:

```
MemoryBlock = AllocMem(ByteSize, Requirements);
    D0            -198      D0            D1
ULONG *MemoryBlock;
ULONG ByteSize;
ULONG Requirements;
```

Description: This function allocates memory from the system memory list. This memory must meet the given requirements and have the desired size.

Parameters:
ByteSize: Size of the requested memory block in bytes.
Requirements: `Flags` value that specifies memory attributes.

Result:
MemoryBlock: Pointer to the allocated memory block. This parameter contains zero if an error occurs.

See Also: `FreeMem()`, `Allocate()`, `AllocEntry()`, `AllocAbs()`, `AllocRemember()`

AllocAbs — Allocates absolute memory

Syntax:

```
MemoryBlock = AllocAbs(ByteSize, Location);
     D0           -204      D0        A1
ULONG *MemoryBlock;
ULONG ByteSize;
ULONG Location;
```

Description: This function attempts to allocate a memory range of specific size in a specific location.

Parameters:
ByteSize: Size of the required memory range in bytes.
Location: Basis address of the memory range.

Result:
MemoryBlock: Pointer to the allocated memory range. This parameter contains zero if an error occurs.

See Also: `FreeMem(),AllocMem(),AllocEntry(),Allocate(), AllocRemember()`

FreeMem — Frees known memory range

Syntax:
```
FreeMem(MemoryBlock, ByteSize);
  -210       A1          D0
ULONG *MemoryBlock;
ULONG ByteSize;
```

Description: This function releases the memory range allocated at the specific location. This range can be accessed again.

Parameters:
MemoryBlock: Pointer to the memory range.
ByteSize: Size of the memory range in bytes.

See Also: `AllocMem(), FreeEntry(), FreeRemember(), Deallocate()`

AvailMem — Finds available memory range

Syntax:
```
Size = AvailMem(Requirements);
 D0     -216      D1
ULONG Size;
ULONG Requirements;
```

Description: This function searches the system list for the largest memory range that has the necessary requirements.

Parameter: Requirements: `Flags` value that characterizes the memory range.

Result: Size: Memory size in bytes.

AllocEntry — Allocates multiple memory ranges

Syntax:
```
MemList = AllocEntry(Entry);
   D0       -222      A0
struct MemList *MemList;
struct MemList Entry;
```

Description: This function attempts to allocate all memory stated in the memory list. Then it returns a pointer to a memory list in which all of the addresses are entered.

Parameter: Entry: Pointer to a `memlist` structure which lists all memory ranges' features and size.

Result: MemList: List containing all memory range addresses.

See Also: `FreeEntry(),AllocMem(),AllocAbs(),Allocate(), AllocRemember()`

```
struct MemList <exec/memory.h>
{
0x00  00  struct Node ml_Node;  /* Node to link more
                                   MemLists */
0x0E  14  UWORD ml_NumEntries;  /* number of entries in this
                                   Structure */
0x10  16  struct MemEntry ml_ME[1];  /* the first entry */
0x18  24
};
ml_me ml_ME
Memory_Types:
MEMF_PUBLIC (1L<<0)   /* memory adds more tasks for use */
MEMF_CHIP (1L<<1)     /* memory can be addressed from certain
                         chips */
MEMF_FAST (1L<<2)     /* the memory cannot be addressed through
                         the certain chips, through which access
                         is faster */
MEMF_CLEAR (1L<<16)   /* the memory should be erased at the same
                         time it is being occupied */
MEMF_LARGEST (1L<<17)  /* the largest memory region with the
                          given criteria is searched for */
MEM_BLOCKSIZE 8L  /* Minimum size of a memory region in bytes */
MEM_BLOCKMASK 7L
struct MemEntry <exec/memory.h>
{
union
   {
   ULONG meu_Reqs; /* Type of the memory */
   APTR meu_Addr;  /* Address of the memory region */
   }
0x00  me_Un;
0x04  04  ULONG me_Length;  /* Length of the memory region */
0x08  08
};
```

Definition:

```
me_un me_Un
me_Reqs me_Un.meu_Reqs
me_Addr me_Un.meu_Addr
```

FreeEntry — Frees multiple memory ranges

Syntax:

```
FreeEntry(Entry);
  -228     A0
struct MemList *Entry;
```

Description: This function restores all of the memory ranges to the list.

Parameter: Entry: Pointer to list containing all memory ranges.

See Also: `AllocEntry(),FreeMem(),Deallocate(), FreeRemember()`

6.1.4 List management functions

Insert	**Inserts node in list**

Syntax:

```
Insert(List, Node, Predecessor);
-234    A0    A1      A2
struct List *List;
struct Node *Node;
struct Node *Predecessor;
```

Description: This function inserts a new node in the list, following the specified node.

Parameters:

List:	Pointer to the head of the list.
Node:	Pointer to the node structure to be inserted in the list.
Predecessor:	Pointer to the node preceding `Node's` insertion point.

See Also: `Remove(),AddHead(),RemHead(),AddTail(),RemTail()`

```
struct List <exec/lists.h>
{
0x00  00  struct Node *lh_Head;  /* pointer to head node of the
                                    list */
0x04  04  struct Node *lh_Tail;  /* pointer to the last
                                    node of the list */
0x08  08  struct Node *lh_TailPred;  /* pointer to the previous
                                        node of list */
0x0C  12  UBYTE lh_Type;  /* Type of this Node */
0x0D  13  UBYTE l_pad;  /* unused full byte */
0x0E  14
};
struct MinList <exec/lists.h>
{
0x00  00  struct MinNode *mlh_Head;  /* pointer to your head
                                        node of the list */
0x04  04  struct MinNode *mlh_Tail;  /* pointer to the last
                                        Node of the list */
0x08  08  struct MinNode *mlh_TailPred;  /* pointer to the
                                        previous Node of list */
0x0C  12
};
struct Node <exec/nodes.h>
{
0x00  00  struct Node *ln_Succ;  /* following Node */
0x04  04  struct Node *ln_Pred;  /* previous Node */
0x08  08  UBYTE ln_Type;  /* Node Type */
0x09  09  BYTE ln_Pri;  /* Node Priority */
0x0A  10  char *ln_Name;  /* pointer to a string */
0x0E  14
};
```

```
struct MinNode <exec/nodes.h>
{
0x00  00  struct MinNode *mln_Succ;  /* following Node */
0x04  04  struct MinNode *mln_Pred;  /* previous Node */
0x08  08
};
Node_Type:
NT_UNKNOWN 0L
NT_TASK 1L
NT_INTERRUPT 2L
NT_DEVICE 3L
NT_MSGPORT 4L
NT_MESSAGE 5L
NT_FREEMSG 6L
NT_REPLYMSG 7L
NT_RESOURCE 8L
NT_LIBRARY 9L
NT_MEMORY 10L
NT_SOFTINT 11L
NT_FONT 12L
NT_PROCESS 13L
NT_SEMAPHORE 14L
NT_SIGNALSEM 15L
```

AddHead — Inserts new node at head of list

Syntax:

```
AddHead(List, Node);
 -240    A0    A1
struct List *List;
struct Node *Node;
```

Description: This function inserts a new node in the head of the list.

Parameters:

List: Pointer to the list.
Node: Pointer to the node structure that should be inserted.

See Also: `Insert(),Remove(),RemHead(),AddTail(),RemTail()`

AddTail — Inserts new node at end of list

Syntax:

```
AddTail(List, Node);
 -246    A0    A1
struct List *List;
struct Node *Node;
```

Description: This function inserts a new node at the end of the list.

Parameters:

List: Pointer to the list.
Node: Pointer to the node structure that should be inserted.

See Also: `Insert(),Remove(),AddHead(),RemHead(),RemTail()`

Remove — Removes node from list

Syntax:

```
Remove(Node);
-252   A1
struct Node *Node;
```

Description: This function removes a node from a list.

Parameter: Node: Pointer to the node structure that should be removed.

See Also: `Insert(),Remove(),AddHead(),AddTail(),RemTail()`

RemHead — Removes first node from list

Syntax:

```
Node = RemHead(List);
D0     -258     A0
struct List *List;
```

Description: This function removes the head (first) node from a list.

Parameter: List: Pointer to the list from which the head node should be removed.

See Also: `Insert(),Remove(),AddHead(),AddTail(),RemTail()`

RemTail — Removes last node from list

Syntax:

```
Node = RemTail(List);
 D0    -264     A0
struct List *List;
```

Description: This function removes the tail (last) node from a list.

Parameter: List: Pointer to the list from which the tail node should be removed.

See Also: `Insert(),Remove(),AddHead(),RemHead(),AddTail()`

Enqueue — Inserts node in list based in priority

Syntax:

```
Enqueue(List, Node);
 -270     A0   A1
struct List *List;
struct Node *Node;
```

Description: This function inserts the given node in the list. The priority of the node determines the position of insertion. The higher the priority, the closer to the head of the list the node is inserted.

Parameters: List: Pointer to the list in which the node should be inserted
Node: Pointer to the node to be inserted.

FindName **Searches for node with matching name**

Syntax:

```
Node = FindName(List, Name);
 D0     -276    A0    A1
struct Node *Node;
struct List *List;
char *Name;
```

Description: This function searches the specified list for a node of the same name. A pointer to the found node structure is returned.

Parameters: List: Pointer to the list that should be searched through.
Name: Pointer to a string that identifies the node.

Result: Node: Pointer to the found node, or zero if an error occurs.

6.1.5 Task functions

AddTask **Adds task to system**

Syntax:

```
AddTask(Task, InitPC, FinalPC);
 -282    A1    A2      A3
struct Task *Task;
ULONG InitPC;
ULONG FinalPC;
```

Description: This function installs a new task in the system.

Parameters: Task: Pointer to an initialized task structure.
InitPC: Starting address of the task.
FinalPC: Return address of the task, or zero for the system routine `FinalPC`.

See Also: `RemTask()`, `FindTask()`

```
extern struct Task <exec/tasks.h>
{
0x00  00  struct Node tc_Node; /* Node for the chaining of
                                  multiple tasks */
0x0E  14  UBYTE tc_Flags;      /* Flags value */
0x0F  15  UBYTE tc_State;      /* Status */
0x10  16  BYTE tc_IDNestCnt;   /* number of connected intr */
0x11  17  BYTE tc_TDNestCnt;   /* number of connected Tasks*/
0x12  18  ULONG tc_SigAlloc;   /* signal provided for task*/
0x16  22  ULONG tc_SigWait;    /* Signal that was waited for */
0x1A  26  ULONG tc_SigRecvd;   /* Signal that was received */
```

```
0x1E  30  ULONG tc_SigExcept;  /* Signal that was taken out */
0x22  34  UWORD tc_TrapAlloc;  /* Traps provided for this task*/
0x24  36  UWORD tc_TrapAble;   /* Traps that are allowed */
0x26  38  APTR tc_ExceptData;  /* pointer to the data that is
                                  taken out */
0x2A  42  APTR tc_ExceptCode;  /* Programmcode code that is
                                  taken out */
0x2E  46  APTR tc_TrapData;    /* pointer to data for the
                                  traps*/
0x32  50  APTR tc_TrapCode;    /* pointer to the program code
                                  for the Traps */
0x36  54  APTR tc_SPReg;       /* pointer to the stack of this
                                  task */
0x3A  58  APTR tc_SPLower;     /* bottom limit of the stack
                                  memory */
0x3E  62  APTR tc_SPUpper;     /* top limit of the stack memory
                                  + 2 */
0x42  66  VOID (*tc_Switch)(); /* pointer to the switch routine:
                                  Task is disengaged from the
                                  CPU */
0x46  70  VOID (*tc_Launch)(); /* pointer to the switch routine:
                                  Task is assigned by the CPU */
0x4A  74  struct List tc_MemEntry; /* pointer to the memory list
                                      with the memory that is
                                      provided for this task */
0x58  88  APTR tc_UserData;    /* pointer to the data of the
                                  users of this Task */
0x5C  92
};
Task_Bytes:
TB_PROCTIME 0L       /* the task has Processor time */
TB_STACKCHK 4L       /* the stack is examined */
TB_EXCEPT 5L        /*the task is excluded from the processor */
TB_SWITCH 6L         /* the task is disengaged from the
                        processor */
TB_LAUNCH 7L         /* the task gets processor time again */
Task_Flags:
TF_PROCTIME (1L<<0)  /* See above */
TF_STACKCHK (1L<<4)
TF_EXCEPT (1L<<5)
TF_SWITCH (1L<<6)
TF_LAUNCH (1L<<7)
Task_State:
TS_INVALID 0L   /* invalid task */
TS_ADDED 1L     /* the task is inserted in the list */
TS_RUN 2L       /* the task is running at the moment */
TS_READY 3L     /*the task is ready to run with the processor */
TS_WAIT 4L      /* the task waits for a signal */
TS_EXCEPT 5L    /* the task is disengaged from the processor */
TS_REMOVED 6L   /*the task is removed */
Signal_Byte:
SIGB_ABORT 0L
SIGB_CHILD 1L
SIGB_BLIT 4L
SIGB_SINGLE 4L
SIGB_DOS 8L
Signal_Flag:
SIGF_ABORT (1L<<0)
SIGF_CHILD (1L<<1)
SIGF_BLIT (1L<<4)
SIGF_SINGLE (1L<<4)
SIGF_DOS (1L<<8)
```

RemTask — Removes task from system

Syntax:

```
RemTask(Task);
  -288   A1
struct Task *Task;
```

Description: This function removes the given task from the system. All previous accesses to resources must be returned as well.

Parameter:

Task: Pointer to thė task structure, or zero for the separate task.

See Also: `AddTask(),FindTask()`

FindTask — Searches task for name

Syntax:

```
Task = FindTask(Name);
 D0      294     A1
struct Task *Task;
char *Name;
```

Description: This function searches the system list for the task of the same name. The separate task is searched with a null pointer.

Parameter:

Name: Pointer to the search task's name.

Result:

Task: Pointer to the found task, or zero if an error occurs.

See Also: `AddTask(),RemTask()`

SetTaskPri — Sets task priority

Syntax:

```
OldPriority = SetTaskPri(Task, Priority);
    D0          -300       A1      D0
BYTE OldPriority;
struct Task *Task;
BYTE Priority;
```

Description: This function assigns the given task a new priority. The system then re-calculates the times assigned to each task. The task with the highest priority is assigned to the processor.

Parameters:

Task: Pointer to the task.
Priority: New priority value of the task.

Result:

OldPriority: Old priority value of the task.

SetSignal — Defines task's signal status

Syntax:

```
OldSignals = SetSignal(NewSignals, SignalSet);
    D0          -306      D0           D1
ULONG OldSignals;
ULONG NewSignals;
ULONG SignalSet;
```

Description: This function assigns a new signal arrangement to the task. This is calculated from the mask and the new signals. The program gets the old signal arrangement back.

Parameters:
NewSignal: New signal value.
SignalSet: Masks from which the signals should be changed.

Result:
OldSignal: Old signal value.

See Also: `AllocSignal(),FreeSignal()`

SetExcept — Sets certain signals as exception triggers

Syntax:

```
OldSignals = SetExcept(NewSignals, SignalSet);
    D0         -312       D0           D1
ULONG OldSignals;
ULONG NewSignals;
ULONG SignalSet;
```

Description: This function defines which signals can generate an exception.

Parameters:
NewSignals: New exception signals.
SignalSet: Mask which specifies the signal bits to be changed.

Result:
OldSignals: Old exception signals.

Wait — Waits for signal(s)

Syntax:

```
Signals = Wait(SignalSet);
   D0     -318    D0
ULONG Signals;
ULONG SignalSet;
```

Description: This function waits until one or more signals occur. The task is set in the `Wait` status, thus requiring no processor time.

Parameter:
SignalSet: Mask containing the desired signal(s).

Result:
Signals: Value received that the signal contains.

See Also: `Signal()`

Signal — Signals report to another task

Syntax:

```
Signal(Task, SignalSet);
-324   A1     D0
struct Task *Task;
ULONG SignalSet;
```

Description: This function sends a signal to another task. This can be taken from the `Wait` status, thus requiring no processor time.

Parameters:

Task: Task in which the signal should be sent.
SignalSet: Mask that contains all of the set bits.

See Also: `Wait()`

AllocSignal — Allocates signal bit

Syntax:

```
SignalNum = AllocSignal(SignalNum);
   D0          -330        D0
LONG SignalNum;
LONG SignalNum;
```

Description: This function attempts to allocate a signal of a task. When you know the number of a free signal, this function can search for a free signal. The function returns a value of -1 back when no signal is free, or when the desired signal was not free.

Parameter: SignalNum: Signal number.

Result: SignalNum: Number of the allocated signals. `SignalNum` contains -1 if no signal was free.

See Also: `FreeSignal()`

FreeSignal — Frees signal bit

Syntax:

```
FreeSignal(SignalNum);
   -336       D0
LONG SignalNum;
```

Description: This function frees a signal allocated by `AllocSignal()`.

Parameter: SignalNum: Number of the signal to be freed.

See Also: `AllocSignal()`

AllocTrap **Allocates processor trap**

Syntax:
```
TrapNum = AllocTrap(TrapNum);
   D0         -342      D0
LONG TrapNum;
LONG TrapNum;
```

Description: This function allocates one of 68000 trap instructions. Values for `TrapNum` can range from -1 to +15. If `TrapNum` = -1, the system allocates the next free trap.

Parameter: TrapNum: Trap number to allocate.

Result: TrapNum: Number of the allocated trap, or -1 if no free traps exist.

See Also: `FreeTrap()`

FreeTrap **Frees processor trap**

Syntax:
```
FreeTrap(TrapNum);
 -348      D0
LONG TrapNum;
```

Description: This function frees a trap allocated by `AllocTrap()`.

Parameter: TrapNum: Trap number to free.

See Also: `AllocTrap()`

6.1.6 Message functions

AddPort **Adds new message port to system**

Syntax:
```
AddPort(Port);
  -354   A1
struct MsgPort *Port;
```

Description: This function inserts a new message port in the system list. The name and the priority of the structure are initialized so the port can sort correctly, and so all of the other tasks can search for the port in which the name is used.

Parameter: Port: Pointer to a half-initialized `MsgPort` structure.

See Also: `RemPort()`, `FindPort()`

```
struct MsgPort <exec/ports.h>
{
0x00  00  struct Node mp_Node;  /* for connecting a constructed
                                   node */
0x0E  14  UBYTE mp_Flags;       /* Flags for the announcement of
                                   the arrival of the reports */
0x0F  15  UBYTE mp_SigBit;      /* is the number of the signal
                                   bit that is assigned to this
                                   port */
0x10  16  struct Task *mp_SigTask; /* pointer to the task
                                     that should be signalled */
0x14  20  struct List mp_MsgList; /* head of the list of all of
                                     the reports */
0x22  34
};
mp_SoftInt mp_SigTask
mp_Flags:
PF_ACTION 3L
PA_SIGNAL 0L
PA_SOFTINT 1L
PA_IGNORE 2L
```

RemPort — Removes message port from system

Syntax:

```
RemPort(Port)
 -360    A1
struct MsgPort *Port;
```

Description: This function removes a port from the system list. This port is no longer accessible through a pointer, and the `FindPort()` function no longer finds this port. After calling this function the memory should be freed again.

Parameter: Port: Pointer to a message port.

See Also: `AddPort()`, `FindPort()`

PutMsg — Puts message in message port

Syntax:

```
PutMsg(Port, Message);
 -366   A0    A1
struct MsgPort *Port;
struct Message *Message;
```

Description: This function places a message in the specified port. A pointer to the message should be used, instead of memory reallocation.

Parameters:

Port: Pointer to the message port.
Message: Pointer to the message structure.

See Also: `GetMsg()`, `ReplyMsg()`

```
struct Message <exec/ports.h>
{
0x00  00  struct Node mn_Node;  /* Node to tie together multiple
                                   Messages  */
0x0E  14  struct MsgPort *mn_ReplyPort;  /* pointer to the port
                                           in which the report
                                           is sent when the
                                           answer follows */
0x12  18  UWORD mn_Length;  /* Byte number of the report */
0x14  20
};
```

GetMsg — Gets message from message port

Syntax:

```
Message = GetMsg(Port)
   D0      -372   A0
struct Message *Message;
struct MsgPort *Port;
```

Description: This function gets the first available message from the message port list. If no message currently exists in the port, the function returns a value of zero.

Parameter: Port: Pointer to the message port.

Result: Message: Pointer to the first message of the message port. A zero is returned if no message exists there.

See Also: `PutMsg(),ReplyMsg(),WaitPort()`

ReplyMsg — Puts message in reply port

Syntax:

```
ReplyMsg(Message);
  -378     A1
struct Message *Message;
```

Description: This function places a received message in the message's reply port, indicating that the receiver of the message has processed it. The task sent by the report can then free the memory.

Parameter: Message: Pointer to the message structure.

See Also: `PutMsg(),GetMsg(),WaitPort()`

WaitPort — Waits until report appears in given port

Syntax:

```
Message = WaitPort(Port);
   D0       -384    A0
struct Message *Message;
struct Port *Port;
```

Description: This function waits until a report appears in the given port. If a message exists, the function returns the pointer to this message.

Parameter: Port: Pointer to the message port.

Result: Message: Pointer to the message structure received.

See Also: `GetMsg(),PutMsg(),ReplyMsg()`

FindPort **Searches for message port**

Syntax:

```
Port = FindPort(Name)
 D0      -390    A1
struct MsgPort *Port;
char *Name;
```

Description: This function searches the system list for the port of the same name. If the name is found, a pointer is returned. A zero is returned if the name is not found.

Parameter: Name: Pointer to the name of the port to search.

Result: Port: Pointer to the port, or zero.

6.1.7 Library functions

AddLibrary **Adds library to system**

Syntax:

```
AddLibrary(Library);
   -396      A1
struct Library *Library;
```

Description: This function inserts a new library in the system list. The library can then be accessed by any task. The function calculates the checksum of the library entries.

Parameter: Library: Pointer to a previously initialized library structure.

See Also: `MakeLibrary(),RemLibrary(),OpenLibrary(),CloseLibrary()`

```
extern struct Library <exec/libraries.h>
{
0x00  00  struct Node lib_Node;  /* node to tie the library into
                                    the system */
0x0E  14  UBYTE lib_Flags;      /* Flags s.u. */
0x0F  15  UBYTE lib_pad;        /* unused fill byte */
0x10  16  UWORD lib_NegSize;    /* size of the vector table in
                                   bytes */
0x12  18  UWORD lib_PosSize;    /* size of the data (also in
                                   bytes) */
0x14  20  UWORD lib_Version;    /* Version number of library */
```

```
0x16  22  UWORD lib_Revision; /* revision number */
0x18  24  APTR lib_IdString;  /* pointer to an identification
                                 text */
0x1C  28  ULONG lib_Sum;      /* checksumof the library */
0x20  32  UWORD lib_OpenCnt;  /* counter that counts how often
                                 this Library was accessed */
0x22  34
};
Lib_Flags:
LIBF_SUMMING (1L<<0)  /* the checksum was calculated from a used
                         task */
LIBF_CHANGED (1L<<1)  /* one or more of the functions of the
                         library were changed */
LIBF_SUMUSED (1L<<2)  /* one checksum should release reset */
LIBF_DELEXP (1L<<3)   /* the library should be closed, but
                         another task is still using it: Wait
                         status */
Definition:
lh_Node lib_Node
lh_Flags lib_Flagslh_pad lib_pad
lh_NegSize lib_NegSize
lh_PosSize lib_PosSize
lh_Version lib_Version
lh_Revision lib_Revision
lh_IdString lib_IdString
lh_Sum lib_Sum
lh_OpenCnt lib_OpenCnt
LIB_VECTSIZE 6L
LIB_RESERVED 4L
LIB_BASE (-LIB_VECTSIZE)
LIB_USERDEF (LIB_BASE-(LIB_RESERVED*LIB_VECTSIZE))
LIB_NONSTD (LIB_USERDEF)
lib_Vectors:
LIB_OPEN (-6L)
LIB_CLOSE (-12L)
LIB_EXPUNGE (-18L)
LIB_EXTFUNC (-24L)
```

CloseLibrary — Closes library

Syntax:

```
CloseLibrary(Library);
    -414      A1
struct Library *Library;
```

Description: This function closes a library to access from a task.

Parameter: Library: Pointer to the library node.

See Also: OpenLibrary()

MakeFunctions — Makes function table

Syntax:

```
TableSize = MakeFunctions(Target, FunctionArray, FuncDispBase);
    D0          -90          A0          A1             A2
ULONG TableSize;
ULONG Target;
ULONG *FunctionArray;
ULONG FuncDispBase;
```

Description: This function creates a function jump table from a table containing function addresses. Libraries, devices and resources require this table, usually calculated using absolute jumps. Relative tables can also be created, using `FuncDispBase`.

Parameters:
- Target: Function jump table address.
- FunctionArray: Pointer to table containing the addresses.
- FuncDispBase: Pointer to basis address to which all of the addresses should be relatively calculated, or zero.

Result: TableSize: Table size in bytes.

MakeLibrary — Creates library

Syntax:

```
Library = MakeLibrary(FuncInit, StructInit, LibInit,
    D0        -84        A0          A1         A2
                              DataSize, CodeSize);
                                  D0       D1
ULONG *Library;
ULONG *FuncInit;
ULONG *StructInit;
ULONG *LibInit;
ULONG DataSize;
ULONG *CodeSize;
```

Description: This function makes an entire library. For this the vector table and a data list are combined. In addition, the routine allocates enough memory, and starts an existing initialization routine.

Parameters:
- FuncInt: Pointer to a table containing all of the table jumps.
- StructInt: Pointer to an `InitStruct` data list.
- LibInt: Address of initialization routine.
- DataSize: Size of this library's data range.
- CodeSize: Pointer to a segment list.

Result: Library: Pointer to a library.

See Also: `InitStruct()`

RemLibrary — Removes library

Syntax:

```
Error = RemLibrary(Library);
  D0      -402       A1
LONG Error;
struct Library *Library;
```

Description: This routine removes a library from the system list. Once removed, this library can no longer be opened through `OpenLibrary()`.

Parameter: Library: Pointer to a library node.

Result: Error: Zero if no error occurs.

See Also: `AddLibrary(),MakeLibrary()`

OldOpenLibrary — absolute OpenLibrary

Syntax:

```
Library = OldOpenLibrary(LibName);
  D0          -408        A1
struct Library *Library;
char *LibName;
```

Description: This function is the old version of the `OpenLibrary()` function. The version number of the library that should be opened is not checked. This function is implemented only so older versions of source texts can be complied and executed without changes.

Parameter: LibName: Pointer to name of the library.

Result: Library: Pointer to opened library; returns zero if an error occurs.

See Also: `OpenLibrary(),CloseLibrary()`

OpenLibrary — Opens library

Syntax:

```
Library = OpenLibrary(LibName, Version);
  D0         -552        A1       D0
struct Library *Library;
char *LibName;
LONG Version;
```

Description: This function opens a library and its functions for access.

Parameters:
LibName: Name of the library to open.
Version: Version number of the library to be opened.

Result: Library: Pointer to the opened library; returns zero if an error occurs.

See Also: `OldOpenLibrary(),CloseLibrary()`

SetFunction — Sets library function vector

Syntax:

```
OldFunct = SetFunction(Library, Offset, FunctEntry);
  D0         -420        A1       A0       D0
LONG OldFunct;
struct Library *Library;
LONG Offset;
ULONG FunctEntry;
```

Description: This routine changes a function of the library by replacing the old address with the new in the offset.

Parameters:
- Library: Pointer to the library in which a function should be changed.
- Offset: Offset of the function to be changed.
- FunctEntry: Pointer to the new function.

Result: OldFunct: Pointer to the old function in the offset.

SumLibrary — **Computes/views library checksum**

Syntax:

```
SumLibrary(Library);
    -426     A1
struct Library *Library;
```

Description: This function calculates the checksum of the library. This can verify whether the library has been illegally tampered with.

Parameter: Library: Pointer to the library.

Explanation: When an error occurs, the user receives an Alert.

6.1.8 Device functions

AddDevice — **Adds device to system**

Syntax:

```
AddDevice(Device);
    -432    A1
struct Device *Device;
```

Description: This function adds a new device to the system. Any program can use this device.

Parameter: Device: Pointer to an initialized device node.

See Also: `RemDevice(),OpenDevice(),CloseDevice()`

```
struct Device <exec/devices.h>
{
0x00  00  struct Library dd_Library;  /* Device Structure =
                                         Library Structure */
0x22  34
};
```

RemDevice — Removes device from system

Syntax:

```
Error = RemDevice(Device);
 D0       -438      A1
LONG Error;
struct Device *Device;
```

Description: This function removes a device from the system. The device cannot be accessed or opened by name.

Parameter: Device: Pointer to an existing device node.

Result: Error: Returns zero if no error occurs during execution.

See Also: AddDevice(),OpenDevice(),CloseDevice()

OpenDevice — Opens device

Syntax:

```
Error = OpenDevice(DevName, Unit, IORequest, Flags);
 D0        -444      A0       D0      A1        D1
LONG Error;
char *DevName;
ULONG Unit;
struct IORequest *IORequest;
ULONG Flags;
```

Description: This function opens the specified device and initializes the given I/O request block.

Parameters:

DevName:	Pointer to the device name.
Unit:	Device dependent unit number.
IORequest:	I/O request block to be filled with data.
Flags:	Mode setting flags (not always used).

Result: Error: Returns zero if no error occurs during execution.

See Also: AddDevice(),RemDevice(),CloseDevice()

CloseDevice — Closes device

Syntax:

```
CloseDevice(IORequest);
   -450       A1
struct IORequest *IORequest;
```

Description: This function closes an open device. Access ends and the `IORequest` structure is freed for further use.

Parameter: IORequest: Pointer to an `IORequest` structure.

See Also: AddDevice(),RemDevice(),OpenDevice()

DoIO — Executes I/O command

Syntax:

```
Error = DoIO(IORequest);
  D0    -456    A1
LONG Error;
struct IORequest *IORequest;
```

Description: This function assigns a device that executes the given command in the `IORequest` structure. The command accesses the task as long as necessary.

Parameter: IORequest: Pointer to an initialized `IORequest` structure.

Result: Error: Returns zero if no error occurs.

See Also: `SendIO(),WaitIO()`

```
struct IORequest <exec/io.h>
{
0x00  00  struct Message io_Message;/* Message Structure */
0x0E  14  struct Device *io_Device; /* pointer to the device to
                                       be addressed */
0x12  18  struct Unit *io_Unit;   /* pointer to the Unit
                                     (which has a different
                                     meaning with each device) */
0x16  22  UWORD io_Command;          /* Device command */
0x18  24  UBYTE io_Flags;            /* Device dependent flags */
0x19  25  BYTE io_Error;             /* Device dependent error
                                        message */
0x1A  26
};
Unit_Flags:
UNITF_ACTIVE (1L<<0)                 /* Unit is addressable */
UNITF_INTASK (1L<<1)                 /* Unit is working */
struct Unit <exec/devices.h>
{
0x00  00  struct MsgPort *unit_MsgPort;  /* pointer to the
                                            message port of the
                                            Unit */
0x04  04  UBYTE unit_flags;
0x05  05  UBYTE unit_pad;
0x06  06  UWORD unit_OpenCnt;        /* number of the inquiry
                                        opened */
0x08  08
};
I/O-Errors:
IOERR_OPENFAIL -1L       /* has not failed */
IOERR_ABORTED -2L        /* was aborted */
IOERR_NOCMD -3L          /* no existing command */
IOERR_BADLENGTH -4L      /* falsche Länge */
struct IOStdReq <exec/io.h>
{
0x00  00  struct Message io_Message;  /* Message Structure */
0x0E  14  struct Device *io_Device;   /* pointer to the device
                                         to be addressed */
0x12  18  struct Unit *io_Unit; /* pointer to the Unit (which
                                   has a different meaning for
                                   each device) */
```

```
0x16  22  UWORD io_Command;     /* Device command */
0x18  24  UBYTE io_Flags;       /* Device dependent flags */
0x19  25  BYTE io_Error;        /* Device dependent error
                                   message */
0x1A  26  ULONG io_Actual;      /* number of the data bytes sent
                                   up to now */
0x1E  30  ULONG io_Length;     /* number of bytes to transfer */
0x22  34  APTR io_Data;         /* pointer to the data buffer */
0x26  38  ULONG io_Offset;      /* Variable for the block
                                   structured devices */
0x2A  42
};
io_Command:
DEV_BEGINIO (-30L)
DEV_ABORTIO (-36L)
io_Fags:
IOB_QUICK 0L
IOF_QUICK (1L<<0)
io_Command:
CMD_INVALID 0L
CMD_RESET 1L
CMD_READ 2L
CMD_WRITE 3L
CMD_UPDATE 4L
CMD_CLEAR 5L
CMD_STOP 6L
CMD_START 7L
CMD_FLUSH 8L
CMD_NONSTD 9L
```

SendIO — Sends I/O command

Syntax:

```
SendIO(IORequest);
 -462     A1
struct IORequest *IORequest;
```

Description: This function assigns a device to execute the command given in the `IORequest` structure. It returns to program execution, instead of waiting for the I/O command to execute.

Parameter: IORequest: Pointer to the initialized `IORequest` structure.

See Also: `DoIO(),WaitIO()`

CheckIO — Returns IORequest status

Syntax:

```
Result = CheckIO(IORequest);
  D0       -468     A1
struct IORequest *Result
struct IORequest *IORequest;
```

Description: This function determines whether an I/O process begun earlier has finished processing. It either returns the pointer to the request structure, or zero if the request is still executing.

Parameter: IORequest: Pointer to the initialized `IORequest` structure.

Result:	Result:	Returns zero if the I/O request is still executing; returns a pointer to the I/O request block in any other case.

WaitIO **Waits until I/O request finishes processing**

Syntax:

```
Error = WaitIO(IORequest);
  D0      -474     A1
LONG Error;
struct IORequest *IORequest;
```

Description: This function waits until the I/O request is processed.

Parameter: IORequest: Pointer to the initialized `IORequest` structure.

Result: Error: Returns zero if no errors occur during execution.

Warning: If the I/O request is not answered, the function never returns!

See Also: `SendIO(),DoIO()`

AbortIO **Aborts running I/O process**

Syntax:

```
Error = AbortIO(IORequest)
  D0     -480       A1
LONG Error;
struct IORequest *IORequest;
```

Description: This function aborts the specified I/O request. If termination is not successful, the function returns an error message.

Parameter: IORequest: Pointer to the initialized `IORequest` structure.

Result: Error: Returns zero if no errors occur during execution.

See Also: `SendIO(),WaitIO(),DoIO()`

6.1.9 Resource functions

AddResource **Inserts resource in system**

Syntax:

```
AddResource(Resource);
   -486        A1
struct Node *Resource;
```

Description: This function inserts a new resource in the system list. Then it can be accessed by all of the other tasks. The resource should already be addressable before addition to the list.

Parameter: Resource: Pointer to an initialized resource node structure.

See Also: RemResource(),OpenResource()

RemResource **Removes resource from system**

Syntax:

```
RemResource(Resource);
   -492        A1
struct Node *Resource;
```

Description: This function removes a resource from the system list.

Parameter: Resource: Pointer to the resource node.

See Also: AddResource(),OpenResource()

OpenResource **Opens resource to access**

Syntax:

```
Resource = OpenResource(Resname, Version);
   D0          -498        A1       D0
struct Node *Resource
char *ResName;
ULONG Version;
```

Description: This function opens a resource to access. The call returns the pointer to the desired resource. The function first searches to see if a resource with the given name is present. Then the version number of the found resource is compared to the `Version` parameter. If the present resource has an older version number, the resource is not opened.

Parameters:
ResName: Pointer to the desired resource name.
Version: Version number.

Result: Resource: Returns zero if no error occurs during execution, or a pointer to the resource if an error occurs.

6.1.10 Semaphore supported functions

InitSemaphore **Initializes signal semaphore structure**

Syntax:

```
InitSemaphore(SignalSemaphore);
      -558        A0
struct SignalSemaphore *SignalSemaphore;
```

Description: This function initializes a `SignalSemaphore` structure. It initializes pointers and the semaphore counter.

Parameter: SignalSemaphore: Pointer to the `SignalSemaphore` structure.

See Also: `ObtainSemaphore(), ReleaseSemaphore(), Procure()`

```
struct SignalSemaphore <exec/semaphores.h>
{
0x00  00  struct Node ss_Link;
0x0e  14  SHORT ss_NestCount;
0x10  16  struct MinList ss_WaitQueue;
0x1C  28  struct SemaphoreRequest ss_MultipleLink;
0x28  40  struct Task *ss_Owner;
0x2C  44  SHORT ss_QueueCount;
0x2E  46
};
struct Semaphore <exec/semaphores.h>
{
0x00  00  struct MsgPort sm_MsgPort;
0x22  34  WORD sm_Bids;
0x24  36
};
struct SemaphoreRequest <exec/semaphores.h>
{
0x00  00  struct MinNode sr_Link;
0x08  08  struct Task *sr_Waiter;
0x0C  12
};
```

ObtainSemaphore — Allows semaphore access

Syntax:

```
ObtainSemaphore(SignalSemaphore);
      -564           A0
struct SignalSemaphore *SignalSemaphore;
```

Description: This function allows access to a `MsgPort`. If another task is currently accessing the same `MsgPort`, this function waits until the "lock" is released.

Parameter: SignalSemaphore:
Pointer to the initialized `SignalSemaphore` structure.

See Also: `ObtainSemaphoreList(), InitSemaphore(), ReleaseSemaphore(), TransferSemaphore(), AttemptSemaphore(), Procure()`

ReleaseSemaphore — Releases signal semaphore access

Syntax:

```
ReleaseSemaphore(SignalSemaphore);
      -570           A0
struct SignalSemaphore *SignalSemaphore;
```

Description: This function releases the access to a `MsgPort` locked by `ObtainSemaphore()`. All waiting tasks can now access the unlocked `MsgPort`.

Parameter: SignalSemaphore:
Pointer to the initialized `SignalSemaphore` structure.

Warning: Never try to free an already free `SignalSemaphore`!

See Also: `ObtainSemaphore()`, `ObtainSemaphoreList()`, `Procure()`

AttemptSemaphore	**Semaphore access (no task wait)**

Syntax:

```
Success = AttemptSemaphore(SignalSemaphore);
   D0          -576               A0
BOLL Success;
struct SignalSemaphore *SignalSemaphore;
```

Description: This function allows access to a `MsgPort` (similar to `ObtainSemaphore()`), except the task is not placed in `Wait` status if the port is locked to another task. If another task is currently accessing the same `MsgPort`, this function ends and returns control to the task.

Parameter: SignalSemaphore:
Pointer to an initialized `SignalSemaphore` structure.

Result: Success: Returns TRUE if the semaphore can be accessed, and FALSE if another task has locked it.

See Also: `ObtainSemaphore()`, `ObtainSemaphoreList()`, `ReleaseSemaphore()`, `Procure()`

ObtainSemaphoreList	**Allows semaphore list access**

Syntax:

```
ObtainSemaphoreList(List);
     -582            A0
struct SignalSemaphore List;
```

Description: This function allows access to a `MsgPort` (similar to `ObtainSemaphore()`), except that a list can be linked from multiple `SignalSemaphore` structures with the value `ss_Link`.

Parameter: List: Linked list from the `SignalSemaphore` structure.

See Also: `ObtainSemaphore()`, `ReleaseSemaphore()`, `ReleaseSemaphoreList()`, `Procure()`

ReleaseSemaphoreList	**Releases semaphore list access**

Syntax:
```
ReleaseSemaphoreList(List);
         -588       A0
struct SignalSemaphore List;
```

Description: This function releases access to a `MsgPort` (similar to `ObtainSemaphore()`), freeing both the `SignalSemaphore` and the entire linked list.

Parameter: List: Linked list from the `SignalSemaphore` structure.

See Also: `ObtainSempahore()`, `ObtainSemaphoreList()`, `ReleaseSemaphore()`

FindSemaphore	**Finds SignalSemaphore**

Syntax:
```
SignalSemaphore = FindSemaphore(Name);
     D0               -594      A0
struct SignalSemaphore *SignalSemaphore;
char *Name;
```

Description: This function searches the `SignalSemaphore` list and returns a pointer to the first semaphore whose name corresponds with the given name. Remember to assign a name to each semaphore.

Parameter: Name: Pointer to the name of the `SignalSemaphore` to be found.

Result: SignalSemaphore:
Pointer to the `SignalSemaphore` if found, or zero if the `signalsemaphore` could not be found.

AddSemaphore	**Adds signalsemaphore**

Syntax:
```
AddSemaphore(SignalSemaphore);
    -600         A0
struct SignalSemaphore *SignalSemaphore;
```

Description: This function inserts a new semaphore in the system list. The name and the priority should be initialized to ensure proper processing.

Parameter: SignalSemaphore:
Pointer to an initialized `SignalSemaphore` structure.

See Also: `RemSemaphore()`, `InitSemaphore()`, `FindSemaphore()`

RemSemaphore	Removes signalsemaphore

Syntax:

```
RemSemaphore(SignalSemaphore);
     -606         A0
struct SignalSemaphore *SignalSemaphore;
```

Description: This function removes a semaphore that was inserted in the system list by `AddSemaphore()`.

Parameter: SignalSemaphore:
Pointer to existing semaphore in the system list.

See Also: `AddSemaphore(),FindSemaphore()`

6.1.11 Kickstart and memory functions

SumKickData	Calculates Kickstart delta list checksum

Syntax:

```
SumKickData()
-612
```

Description: This function calculates the checksum of the `KickMemPrt` structure and the `KickTagPrt` array. This value is stored in the `KickCheckSum` array of the `ExecBase` structure.

See Also: `InitResident(),FindResident()`

AddMemList	Adds free memory

Syntax:

```
Error = AddMemList(Size, Attributes, Pri, Base, Name);
  D0     -618      D0       D1       D2    A0    A1
LONG Error;
ULONG Size;
UWORD Attributes;
BYTE Pri;
ULONG Base;
char *Name;
```

Description: This function defines a new memory range that is added to the list of free memory.

Parameters:

Size:	Size of new memory region in bytes.
Attributes:	Memory attributes.
Pri:	Priority of memory (-10 = Chip, 0 = Fast).
Base:	Basis address of memory.
Name:	Name entered in the memory header.

Result: Error: Error number.

See Also: `Allocate()`, `AllocMem()`, `AllocEntry()`, `AllocRemember()`

CopyMem — Copies memory

Syntax:

```
CopyMem(Source, Dest, Size);
  -624    A0     A1    D0
ULONG Source, Dest;
ULONG Size;
```

Description: This function copies the given memory range to a new position by the fastest means.

Parameters:

Source: Basis address of the source range.
Dest: Basis address of the destination range.
Size: Size of the memory region to copy in bytes.

See Also: `CopyMemQuick()`

CopyMemQuick — Copies memory quickly

Syntax:

```
CopyMemQuick(Source, Dest, Size);
     -630      A0     A1    D0
ULONG Source, Dest;
ULONG Size;
```

Description: This routine is the improved version of the `CopyMem()` function. It copies faster. The basis addresses and memory size must be given in LONG format, which means only four divisible addresses may be accessed at a time.

Parameters:

Source: Basis address of the source range.
Dest: Basis address of the destination range.
Size: Size of the memory region to copy in bytes.

Warning: System interrupt will occur if you do not follow address conventions.

See Also: `CopyMem()`

6.1.12 Additional Functions

RawDoFmt	Formats character string data

Syntax:

```
RawDoFmt(FormatString, DataStream, PutChProc, PutChData);
  -552       A0            A1          A2          A3
ULONG *FormatString;
ULONG *DataStream;
ULONG PutChProc;
ULONG PutChData;
```

Description: This function displays a string. Numeric values are converted as usual in C. The function needs a pointer to the string, a pointer to the data, the address of a routine that displays each character and the register to which the character code should be transferred.

Parameters:

FormatString: Pointer to a string containing format elements.
DataStream: Pointer to the data to be replaced.
PutChProc: Address of the character display routine.
PutChData: Address register to which the data should be given.

GetCC	Condition codes for compatible processor

Syntax:

```
Conditions = GetCC();
    D0       -528
ULONG Conditions;
```

Description: This function supplies the condition codes to allow compatibility with a 68010 processor.

Result: Condition codes

TypeOfMem	Returns memory attributes

Syntax:

```
Attributes = TypeOfMem(Address);
    D0          -534      A1
ULONG Attributes;
ULONG Address;
```

Description: This function returns the attributes of the specified memory range.

Parameter: Address: Address of memory range.

Result: Attributes: Attributes and type of memory range.

6.2 The DOS library

The DOS library provides the first line of communication with the disk drives or the hard disk. Some DOS functions can be found in the C standard library. C limits the number of files that may be open at the same time, while the DOS library allows as many files open at the same time as memory permits.

If you intend to use your program in conjunction with other programs later, use the C standard functions to make transportation of the program easier. The DOS library also contains functions that allow creation of a new process (loading and starting new programs from disk), pausing a program for a specified time and protecting files from accidental deletion.

This list includes the `DOSBase` structure, in which you'll see different devices. Through that you can determine which drives are connected to the Amiga, because a drive is also a device (DF0:, DF1:, ...).

DOS library functions

6.2.1 Input and output

6.2.2 File management

6.2.1 Input and output

Close **Closes file**

Syntax:

```
Close(File)
-36   D1
struct FileHandle *File;
```

Description: This function closes a file opened by `Open`.

Parameter: File: File specifier.

Comments: You must close any open files before ending a program. Unclosed files stay open until the next reset.

Warning: Do not confuse the DOS `Close` function with the C functions `close()` or `fclose()`. The three functions use three different file specifiers.

See Also: `Open`

Open — Opens file

Syntax:

```
File = Open(Name,Mode)
 D0     -30  D1   D2
struct FileHandle *File;
UBYTE *Name;
LONG Mode;
```

Description: This function opens a file for reading or writing. If the `Mode` is `MODE_OLDFILE`, an existing file opens. `MODE_NEWFILE` creates a new file and deletes an existing file of the same name. `MODE_READWRITE` opens an existing file but allows exclusive access to this file. Exclusive access means that when you open a file with `MODE_READWRITE`, no other task can open this file. The `Mode` parameter has no meaning if you read the file.

Parameters:

Name: Pointer to the filename.

Mode: Tests for mode (`MODE_OLDFILE` for opening existing file, `MODE_READWRITE` for exclusive mode or `MODE_NEWFILE` for creating a new file).

Result:

File: File specifier.

Exception: The function returns a zero if the file cannot be opened. `IoErr` returns the exact error message.

Warning: Do not confuse the DOS `Open` function with the C functions `open()` or `fopen()`. The three functions use three different file specifiers.

Comments: Use `Lock()` if you only want to see if a file exists.

See Also: `Close`, `IoErr`

Read — Reads file data

Syntax:

```
Number = Read(File,Buffer,Length)
  D0     -42  D1     D2      D3
LONG Number;
struct FileHandle *File;
UBYTE *Buffer;
LONG Length;
```

Description: This function reads data from a file into a buffer.

Parameters: File: File specifier.
Buffer: Pointer to the data buffer.
Length: Number of bytes to be read.

Result: Number: Number of bytes actually read.

Exceptions: If `number` = zero, this indicates the end of the file. If `number` < 0, an error occurred. `IoErr` returns the exact error message.

Comments: The value that `IoErr` returns is changed every time by `Read`. When no error is encountered, `IoErr` indicates whether data can still be inserted in the file.

See Also: `Open, Close, Write`

Seek — Sets file pointer

Syntax:

```
oldPosition = Seek(File,Position,Mode)
    D0         -66   D1     D2     D3
LONG oldPosition;
struct FileHandle *File;
LONG Position, Mode;
```

Description: This function sets the file pointer at `position`. The file pointer can be set based on three `mode` options:

`OFFSET_BEGINNING`
: Sets the file pointer to `position` bytes after the beginning of the file. `position` must be a positive value.

`OFFSET_END`
: Sets the file pointer to `position` bytes before the end of the file. `position` must be a negative value.

`OFFSET_CURRENT`
: Sets the file pointer to `position` bytes from the current file position. `position` can be a positive or negative value.

Parameters: File: File specifier.
Position: The number of bytes the file pointer should be moved.
Mode: Movement mode.

Result: oldPosition: Old file pointer position relative to the beginning of the file.

Exceptions: If `oldPosition` = -1, an error occurred. `IoErr` returns the exact error message.

Comments: Calling `Seek` returns the position of the current file pointer:

```
pos = Seek(file,0L,OFFSET_CURRENT);
```

You can easily set the file pointer to the beginning or the end of the file by calling `Seek` with `position` equal to zero and calling `OFFSET_BEGINNING` and `OFFSET_END` as mode.

Write — Writes data to file

Syntax:

```
Number = Write(File,Buffer,Length)
  D0      -48   D1    D2     D3
LONG Number;
struct FileHandle *File;
UBYTE *Buffer;
LONG Length;
```

Description: This function writes data from a buffer into a file.

Parameters:

File:	File specifier.
Buffer:	Pointer to the data buffer.
Length:	Number of bytes that should be written.

Result:

Number:	Number of bytes actually written.

Exceptions: If `number` is negative, an error occurred. `IoErr` returns the exact error message.

See Also: `Open`, `Close`, `Read`

Structures:

```
MODE_READWRITE      1004
MODE_OLDFILE        1005
MODE_NEWFILE        1006

OFFSET_BEGINNING   -1
OFFSET_CURRENT      0
OFFSET_END          1

struct FileHandle <libraries/dosextens.h>
{
0x00  0  struct Message *fh_Link;
0x04  4  struct MsgPort *fh_Port;  /* <> 0 => Interactive */
0x08  8  struct MsgPort *fh_Type;
0x0C 12  LONG fh_Buf;
0x10 16  LONG fh_Pos;
0x14 20  LONG fh_End;
0x18 24  LONG fh_Funcs;
0x1C 28  LONG fh_Func2;
0x20 32  LONG fh_Func3;
0x24 36  LONG fh_Args;
0x28 40  LONG fh_Arg2;
0x2C 44
};
```

6.2.2 File management

CreateDir **Creates new subdirectory**

Syntax:

```
Lock = CreateDir(Name)
 D0        -120    D1
struct FileLock *Lock;
UBYTE *Name;
```

Description: This function creates a new subdirectory within the current directory, if possible. You can create a subdirectory on a floppy or hard disk, but not on CON:, PRT:, or other devices that cannot support subdirectories.

Parameter: Name: Pointer to the directory name. You can also give a path name. If you enter a path, all directories entered in the path (except the one you wish to create, which must be listed last) must already exist.

Result: Lock: BCPL pointer to the lock of the new directory (type `ACCESS_READ`).

Exceptions: If the subdirectory cannot be added, the function returns zero. `IoErr` returns the exact error message.

See Also: `IoErr`, `Lock`, `UnLock`

CurrentDir **Specifies the current directory**

Syntax:

```
oldLock = CurrentDir(Lock)
  D0           -126     D1
struct FileLock *oldLock;
struct FileLock *Lock;
```

Description: This function sets the current directory. All of the relative path names (path names that have either a device name ["DF0:s/StartUp-Sequence"] or a colon (":c/dir") at the beginning) can be the current directory. The CLI command `cd directory` specifies the current directory.

Parameter: Lock: BCPL pointer to the lock representing a directory.

Result: oldLock: BCPL pointer to the lock that represented the previous current directory. If `oldLock` contains a zero, the old directory was the root directory of the boot disk.

See Also: `Lock`, `UnLock`

DeleteFile **Deletes file or directory**

Syntax:

```
ok = DeleteFile(Name)
D0     -72      D1
BOOL OK;
UBYTE *Name;
```

Description: This function deletes the specified file or empty directory.

Parameter: Name: Pointer to the filename.

Result: OK: Returns FALSE if the file or directory cannot be deleted. `IoErr` returns the exact error message.

See Also: `IoErr`

Examine **Reads file/directory information**

Syntax:

```
ok = Examine(Lock,infoBlock)
D0     -102   D1     D2
BOOL ok;
struct FileLock *Lock;
struct FileInfoBlock *infoBlock;
```

Description: This function fills the `FileInfoBlock` with information about the file or directory. The information can be file size, file type or creation date as represented by the lock.

Parameters: Lock: BCPL pointer to the lock of the file or directory about which information is desired.

infoBlock: Pointer to the `FileInfoBlock` to which the information should be written.

Result: OK: Returns FALSE if information cannot be used.

Warning: The `FileInfoBlock` must be at an address that is divisible by four. For optimum memory use, `AllocMem` allocates memory an address that is divisible by eight.

See Also: `ExNext`

ExNext **Reads next directory entry**

Syntax:

```
ok = ExNext(Lock,infoBlock)
D0   -108    D1     D2
BOOL OK;
struct FileLock *Lock;
struct FileInfoBlock *infoBlock;
```

Description: This function reads the entries in a directory or subdirectory, as well as the directory's lock. `Examine` must be called before you call `ExNext`, so that the `FileInfoBlock` is filled with the necessary starting values. Then you call `ExNext` until the function returns FALSE.

Parameters:
- Lock: BCPL pointer to the lock of the specified directory
- infoBlock: Pointer to the `FileInfoBlock` previously initialized by a call to `Examine`.

Result:
- OK: Returns FALSE when an error occurs, or when no more files exist in the directory. `IoErr` returns the exact error message. When no more files were present, `IoErr` displays the `ERROR_NO_MORE_ENTRIES` error message.

Warning: The `FileInfoBlock` must be at an address that is divisible by four. For optimum memory use, `AllocMem` allocates memory an address that is divisible by eight.

Warning: The `FileInfoBlock` must be at an address that is divisible by four. For optimum memory use, `AllocMem` allocates memory an address that is divisible by eight.

Comments: If you don't know exactly what you're searching for (e.g., filename from an external source), the `type` array of the `FileInfoBlock` structure can be selected.

See Also: `Examine`

Info — Reads disk information

Syntax:

```
ok = Info(Lock,parameterBlock)
D0   -114 D1       D2
BOOL OK;
struct FileLock *Lock;
struct InfoData *parameterBlock;
```

Description: This function supplies information about the disk on which the file or directory lies, represented by `Lock`. This information includes the size of the diskette and the number of free and used blocks.

Parameters:
- Lock: BCPL pointer to the lock of a file or a directory on the disk.
- parameterBlock: Pointer to the `InfoData` structure.

Result:
- OK: Returns FALSE if the information cannot be used.

Warning: The `ParameterBlock` must be at an address that is divisible by four. For optimum memory use, `AllocMem` allocates memory an address that is divisible by eight.

ParentDir — Gets lock of higher directory

Syntax:

```
newLock = ParentDir(Lock)
  D0        -210     D1
struct FileLock *newLock;
struct FileLock *Lock;
```

Description: This function returns the lock of the next directory up in the directory structure.

Parameter: Lock: BCPL pointer to the lock.

Result: newLock: BCPL pointer to the lock of the next directory up.

Comments: If `newLock` returns a zero, the next directory is the root directory.

See Also: `CurrentDir`

Rename — Renames file or directory

Syntax:

```
OK = Rename(oldName,newName)
D0     78       D1      D2
BOOL OK;
UBYTE *oldName,*newName;
```

Description: This function renames the file or directory called `oldName` to `newName`. Both names may contain paths (i.e., the file can be placed in another directory using this function).

Parameters:
oldName: (Path) name of the file to be renamed.
newName: (Path) name of the new filename.

Result: OK: Returns FALSE if a file with the same name already exists.

Comments: You can move a file from one directory to another very easily by renaming paths but retaining the original filename.

Warning: You cannot rename devices using this function.

SetComment — Places comments in file

Syntax:

```
OK = SetComment(Name,Comments)
D0      -180      D1     D2
BOOL OK;
UBYTE *Name,Comments;
```

Description: This function places comments in a file. The comments can be a maximum of 80 characters in length and must end with a null byte.

Parameters:
Name: Pointer to the filename.
Comments: Pointer to the comments.

Result:
OK: Returns FALSE if the comments cannot be written (e.g., disk is write protected).

SetProtection	**Sets protection bits in file**

Syntax:

```
OK = SetProtection(Name,Mask)
D0        -186        D1   D2
BOOL ok;
UBYTE *Name;
LONG Mask;
```

Description: This function sets the protection bits of a file or a directory. The bits have the following meanings:

Bit 0: 1 = File not deletable.
Bit 1: 1 = File not executable (applies to program files only).
Bit 2: 1 = File cannot be overwritten.
Bit 3: 1 = File cannot be read.
Bit 4: Archive bit—deleted every time a file is closed after write access.

Only bits 0 and 4 are currently supported by AmigaDOS.

Parameters:
Name: Pointer to the filename.
Mask: Bit mask.

Result:
OK: Returns FALSE when the changes to the protection bits cannot be processed.

Comments: Bit 4 can be used to determine if a file has been changed since the last (read) access, provided that the archive bit can be set after the read access.

Structure:

```
struct FileLock <libraries/dosextens.h>
{
0x00    0 BPTR              fl_Link;   /* Linked list */
0x04    4 LONG              fl_Key;    /* Disk block no */
0x08    8 LONG              fl_Access; /* Access mode */
0x0C   12 struct MsgPort *fl_Task;
0x10   16 BPTR              fl_Volume; /* in device list */
0x14   20
};
Access modes (fl_Access):
ACCESS_READ           -2   /* Read access */
ACCESS_WRITE          -1   /* Write access */
```

```
struct FileInfoBlock <libraries/dos.h>
{
0x00    0 LONG          fib_DiskKey;
0x04    4 LONG          fib_DirEntryType; /* If < 0 => file */
                                    /* If > 0 => directory */
0x08    8 char          fib_FileName[108];
0x74  116 LONG          fib_Protection;
0x78  120 LONG          fib_EntryType;
0x7C  124 LONG          fib_Size; /* Size in bytes */
0x80  128 LONG          fib_NumBlocks; /* no. blocks */
0x84  132 struct        DateStamp fib_Date;
0x90  144 char          fib_Comment[16];
0x104 260
};
Protection-Flags (fib_Protection):
FIBF_ARCHIVE   16
FIBF_READ       8
FIBF_WRITE      4
FIBF_EXECUTE    2
FIBF_DELETE     1

struct InfoData ,libraries/dos.h>
{
0x00    0  LONG         id_NumSoftErrors; /* Number of errors */
0x04    4  LONG         id_UnitNumber;
0x08    8  LONG         id_DiskState;
0x0C   12  LONG         id_NumBlocks;
0x10   16  LONG         id_NumBlocksUsed;
0x14   20  LONG         id_BytesPerBlock;
0x18   24  LONG         id_DiskType;
0x1C   28  BPTR         id_VOlumeNode;
0x20   32  LONG         id_InUse /* 0 if not used */
0x24   36
};
Disk-Status (id_DiskState):
ID_WRITE_PROTECTED 80
ID_VALIDATING      81
ID_VALIDATED       82
Disk-Type(id_DiskType):
ID_NO_DISK_PRESENT       -1
ID_UNREADABLE_DISK      'BAD'
ID_DOS_DISK             'DOS'
ID_NOT_REALLY_DOS      'NDOS'
ID_KICKSTART_DISK      'KICK'
```

6.2.3 File management utility functions

DupLock — **Creates duplicate lock**

Syntax:

```
newLock = DupLock(Lock)
   D0       -96    D1
struct FileLock *newLock;
struct FileLock *Lock;
```

Description: Creates a duplicate of the specified lock. DupLock copies only read locks and locks of type ACCESS_READ because write locks (ACCESS_WRITE) require exclusive access.

Parameter: Lock: BCPL pointer to the read lock.

Result: newLock: BCPL pointer to the copy of the lock.

See Also: Lock, UnLock

Input — Determines file input channel

Syntax:

```
file = Input()
 D0     -54
struct FileHandle *file;
```

Description: This function returns the handle of the standard input channel.

Result: File: File specifier for output.

Comments: The standard input appears in a CLI window. You can redirect this using <Filename. If you want to get input from the standard input device of your program, you must identify the file specifier from the Input function. If you use the C standard functions (e.g., scanf), the C compiler performs this task for you.

See Also: Output

IoErr — Converts error message from last error number

Syntax:

```
Error = IoErr()
  D0     -132
LONG Error;
```

Description: This function returns an error message based on the most recent DOS error. Most of the DOS functions return zero when an error is encountered; IoErr gives the user a text message.

Result: Error: Error code of the last DOS command.

Comments: The DeviceProc function returns a second result value through IoErr.

See Also: Open, Read, ExNext, DeviceProc

IsInteractive — Identifies virtual terminal

Syntax:

```
Status = IsInteractive(Ffile)
  D0          -216        D1
BOOL Status;
struct FileHandle *File;
```

Description: This function determines whether or not to handle a file as a virtual terminal (e.g., CON:).

Parameter: File: File specifier.

Result: Status: Returns TRUE if file is a virtual terminal.

Lock — Determines file lock

Syntax:

```
lock = Lock(Name,AccessMode)
 D0    -84   D1      D2
struct FileLock *Lock;
UBYTE *Name;
LONG AccessMode;
```

Description: This function identifies the lock of a file or directory. The DOS library accesses many files through locks because of the Amiga's multitasking operating system. If you want to work with a file or a directory, a lock ensures that no other task will delete or change the file/directory during your access.

There are two types of locks: locks for reading and locks for writing. A file can have any number of read locks at one time. Only one write lock can exist at a time, provided the file has no read lock attached.

Parameters:

Name: Pointer to the file or directory name.

AccessMode: Access mode of the file. `AccessMode = ACCESS_READ` requests a read lock, while `AccessMode = ACCESS_WRITE` requests a write lock.

Result: Lock: BCPL pointer to the lock.

Exceptions: If the lock cannot be used (e.g., trying to place a write lock on a file on which a read lock already exists), a zero is returned.

Comments: If you want to know if a file or a directory exists, try placing a lock on the file. This is more efficient than trying to open the file. If you want to actually open the file, don't use the `Lock` function—just open the file.

Warning: Release a lock when you're finished with the file. Locked files remain locked until unlocked or until the next reset, and locked files cannot be deleted or edited.

See Also: UnLock, DupLock

Output — Determines file input channel

Syntax:

```
File = Output()
 D0      -60
struct FileHandle *File;
```

Description: This function returns the handle of the standard output channel.

Result: File: File specifier for output.

Comments: The standard output appears in a CLI window. You can redirect this using >Filename. If you want to send output from the standard output device of your program, you must identify the file specifier from the Output function. If you use the C standard functions (e.g., printf), the C compiler performs this task for you.

See Also: Input

UnLock — Releases lock from file or directory

Syntax:

```
UnLock(Lock)
 -90    D1
struct FileLock *Lock;
```

Description: This function frees the lock on a file or a directory placed there by the Lock or DupLock function.

Parameter: Lock: BCPL pointer to the lock.

Warning: Release a lock when you're finished with the file. Locked files remain locked until unlocked or until the next reset, and locked files cannot be deleted or edited.

See Also: Lock, DupLock

Structure:

```
struct FileLock <libraries/dosextens.h>
{
0x00  0  BPTR            fl_Link;   /* Linked list */
0x04  4  LONG            fl_Key;    /* Disk block no. */
0x08  8  LONG            fl_Access; /* Access mode */
0x0C 12  struct MsgPort *fl_Task;
0x10 16  BPTR            fl_Volume; /* in device list */
0x14 20
};
```

```
Access-modes (fl_Access):
ACCESS_READ          -2  /* Read access */
ACCESS_WRITE         -1  /* Write access */
Fehlermeldungen
ERROR_NO_FREE_STORE               103
ERROR_TASK_TABLE_FULL             105
ERROR_LINE_TOO_LONG               120
ERROR_FILE_NOT_OBJECT             121
ERROR_INVALID_RESIDENT_LIBRARY    122
ERROR_NO_DEFAULT_DIR              201
ERROR_OBJECT_IN_USE               202
ERROR_OBJECT_EXISTS               203
ERROR_DIR_NOT_FOUND               204
ERROR_OBJECT_NOT_FOUND            205
ERROR_BAD_STREAM_NAME             206
ERROR_OBJECT_TOO_LARGE            207
ERROR_ACTION_NOT_KNOWN            209
ERROR_INVALID_COMPONENT_NAME      210
ERROR_INVALID_LOCK                211
ERROR_OBJECT_WRONG_TYPE           212
ERROR_DISK_NOT_VALIDATED          213
ERROR_DISK_WRITE_PROTECTED        214
ERROR_RENAME_ACROSS_DEVICES       215
ERROR_DIRECTORY_NOT_EMPTY         216
ERROR_TOO_MANY_LEVELS             217
ERROR_DEVICE_NOT_MOUNTED          218
ERROR_SEEK_ERROR                  219
ERROR_COMMENT_TOO_BIG             220
ERROR_DISK_FULL                   221
ERROR_DELETE_PROTECTED            222
ERROR_WRITE_PROTECTED             223
ERROR_READ_PROTECTED              224
ERROR_NOT_A_DOS_DISK              225
ERROR_NO_DISK                     226
ERROR_NO_MORE_ENTRIES             232
```

6.2.4 Process management

CreateProc — **Creates new process**

Syntax:

```
process = CreateProc(Name,Pri,Segment,StackSize)
  D0          -138      D1    D2   D3        D4
struct Process *process;
UBYTE *Name;
LONG Pri;
BPTR *Segment;
LONG StackSize;
```

Description: This function initializes and accesses a process data structure. The segment list is obtained by calling the `LoadSeg` function.

Parameters:

	Name:	Name of the process (a different name from the filename under which the program is saved on disk).
	Pri:	Task priority. Values for `Pri` may range from -128 to +127, but not within the range from -20 to 20 (values within this range may cause memory conflicts with the operating system).
	Segment:	BCPL pointer to the segment list as stated in `LoadSeg`.
	StackSize:	Program stack size in bytes.

Result: Process: BCPL pointer to the MsgPort of the process data structure.

Exceptions: If the process cannot be started, a zero is returned. `IoErr` returns the exact error message.

See Also: `LoadSeg`, `UnLoadSeg`

DateStamp — Gets current time and date

Syntax:

```
DateStamp(Ptr)
   -192   D1
LONG *Ptr;
```

Description: This function fills three long words with the current time.

Parameter: Ptr: Pointer to a data buffer that must contain at least three long words (12 bytes). The first long word gives the number of days since 1/1/78, the second long word gives the number of minutes since midnight, and the third gives the number of ticks since the full minute (1 tick = 1/60 second).

See Also: Data structure of `DateStamps`

Delay — Waits a certain time span

Syntax:

```
Delay(Ticks)
-198   D1
LONG Ticks;
```

Description: This function pauses a program for the amount of time specified.

Parameter: Ticks: Length of delay (one tick = 1/60 second).

Comments: If you want a program to wait for a certain time span for any reason, use the `Delay` function instead of a wait loop: The `Delay` function is more reliable.

DeviceProc — Tests process ID of a device driver

Syntax:

```
Process = DeviceProc(Name)
   D0          -174    D1
struct Process *Process;
UBYTE *Name;
```

Description: This function tests the device driver that belongs to the specified device.

Parameter: Name: Device name or filename on which the device driver should be tested.

Result: Process: BCPL pointer to the message port of the device driver's process structure.

Exceptions: If the function returns a zero, the device driver cannot be tested. `IoErr` returns the exact error message.

Comments: When it handles the "device" as a file on an inserted diskette, you get a BCPL pointer over `IoErr` to the lock of the directory in which the file is (assuming no error occurred).

Exit — Ends program

Syntax:

```
Exit(errorCode)
-144     D1
LONG errorCode;
```

Description: This function ends the currently running program, and returns the error code if the program was started from the CLI.

Parameter: errorCode: Error type (0 = no error).

Warning: `Exit` ends the program without closing open files, windows or screens, and without freeing locks. Call the `Exit` function only when you are sure that everything open has been closed or released (locks, file identifications, windows, memory, etc.).

WaitForChar — Waits for available character

Syntax:

```
ok = WaitForChar(File,Time)
D0      -204       D1   D2
BOOL OK;
struct FileHandle *File;
LONG Time;
```

Description: This function waits for a character to be entered within a defined time span. The file must be handled as a virtual terminal (CON:—see `IsInteractive`). `WaitForChar` only determines if characters can be inserted. You must then read this using the `Read` function.

Parameters:

File:	File specifier of a virtual terminal.
Time:	Time in microseconds that `WaitForChar` waits for a character until it returns a value.

Result:

OK:	Returns FALSE if no characters were entered within the specified time.

See Also: `IsInteractive`

Structure:

```
struct DateStamp <libraries/dos.h>
{
0x00  0  LONG  ds_Days;   /* days since 1.1.78 */
0x04  4  LONG  ds_Minute; /* Minutes since midnight */
0x08  8  LONG  ds_Tick;   /* number of ticks */
0x0C 12
};
TICKS_PER_SECOND      60
struct Process <libraries/dosextens.h>
{
0x00   0   struct  Task    pr_Task;
0x5C  92   struct  MsgPort pr_MsgPort; /* CreateProc and      */
                                       /* DeviceProc supply a */
                                      /* BCPL pointer to this */
                                       /* Message Port.       */
0x7E 126   WORD    pr_Pad;
0x80 128   BPTR    pr_SegList;
0x84 132   LONG    pr_StackSize;
0x88 136   APTR    pr_GlobVec;
0x8C 140   LONG    pr_TaskNum; /* =0, if not from the CLI*/
0x90 144   BPTR    pr_StackBase;
0x94 148   LONG    pr_Result2;
0x98 152   BPTR    pr_CurrentDir;
0x9C 156   BPTR    pr_CIS;         /* input channel */
0xA0 160   BPTR    pr_COS;         /* output channel */
0xA4 164   APTR    pr_ConsoleTask;
0xA8 168   APTR    pr_FileSystemTask;
0xAC 172   BPTR    pr_CLI;
0xB0 176   APTR    pr_ReturnAddr;
0xB4 180   APTR    pr_PktWait;
0xB8 184   APTR    pr_WindowPtr;
0xBC 188
};
```

A segment is constructed as follows:

```
-4  LONG    Segment length + 8;
 0  BPTR    Next segment (0 if none);
 4  Code
```

You always receive a pointer to the element containing index 0 from `LoadSeg`. The actual code begins 4 bytes later.

6.2.5 Loading programs

Execute — **Executes CLI command**

Syntax:

```
ok = Execute(Command,Input,Output)
D0     -222     D1     D2    D3
BOOL OK;
UBYTE *Command;
struct FileHandle *Input,*Output;
```

Description: This function executes a CLI command as if you had entered this directly in the CLI. Input and output can be redirected. When the input file is unequal to zero, commands are read from the input file after command processing until the end of the file is reached. If the output file is zero, the current window ("*") is used, which may not function properly if the program is started from the Workbench.

Parameters:

Command:	Pointer to the command(s) to be executed.
Input:	File specifier for input (usually 0).
Output:	File specifier for output, (usually the current [CLI] window).

Result: OK: Returns FALSE if it cannot be processed as usual.

LoadSeg — **Loads a program**

Syntax:

```
Segment = LoadSeg(name)
  D0        -150    D1
BPTR Segment;
UBYTE *Name;
```

Description: This function loads the specified program into memory. The program can be started using `CreateProc`.

Parameter: Name: Pointer to the filename of the program.

Result: Segment: BCPL pointer to the first segment.

Exceptions: If the program could not be started (e.g., not an executable program), the function returns a zero and releases memory.

See Also: `UnLoadSeg,CreateProc`

UnLoadSeg **Removes program**

Syntax:

```
OK = UnLoadSeg(Segment)
D0      -156      D1
BOOL OK;
BPTR Segment;
```

Description: This function removes the program previously loaded into memory using the `LoadSeg` function.

Parameter: Segment: BCPL pointer to the segment.

Result: OK: Returns FALSE if an error occurs.

Comments: If you started the program with `CreateProc`, you won't have to free the segment because `CreateProc` automatically removes the segment when the program ends.

See Also: `LoadSeg`

Structures: A segment is constructed as follows:

```
-4  LONG    Length of the segment + 8;
 0  BPTR    Next Segment (0 if none);
 4  Code
```

You always get a pointer to the element with the index of 0 from `LoadSeg`. The actual code begins 4 bytes later.

6.2.6 Internal DOS functions

GetPacket **Gets DOS packet**

Syntax:

```
OK = GetPacket(WaitForIt)
D0      -162      D1
BOOL OK;
BOOL WaitForIt;
```

Description: This function gets a DOS packet sent from another process.

Parameter: WaitForIt: Ready status—registers TRUE while waiting for a DOS packet.

Result: OK: Packet status—returns a zero if no packet exists and `WaitForIt` registers FALSE.

QueuePacket — Sends DOS packet to another process

Syntax:

```
Error = QueuePacket(Packet)
  D0        -168      D1
LONG Error;
struct DosPacket *Packet;
```

Description: Sends a DOS packet to the process stated in the `DOSPacket` structure.

Parameter: Packet: Pointer to the DOS packet to be sent.

Result: Error: Returns zero if an error occurs.

Structures:

```
struct DosPacket <libraries/dosextens.h>
{
0x00  0  struct Message *dp_Link;    /* set to zero */
0x04  4  struct MsgPort *dp_Port;
0x08  8  LONG dp_Type;
0x0C 12  LONG dp_Res1;                         /* result */
0x10 16  LONG dp_Res2;      /* 2. result => IoErr() */
0x14 20  LONG dp_Arg1;
0x18 24  LONG dp_Arg2;
0x1C 28  LONG dp_Arg3;
0x20 32  LONG dp_Arg4;
0x24 36  LONG dp_Arg5;
0x28 40  LONG dp_Arg6;
0x2C 44  LONG dp_Arg7;
0x30 48
};
struct StandardPacket <libraries/dosextens.h>
{
0x00  0  struct Message   sp_Msg;
0x14 20  struct DosPacket sp_Pkt;
0x44 68
};
Packet-type (dp_Type):
ACTION_NIL               0
ACTION_GET_BLOCK         2
ACTION_SET_MAP           4
ACTION_DIE               5
ACTION_EVENT             6
ACTION_CURRENT_VOLUME    7
ACTION_LOCATE_OBJECT     8
ACTION_RENAME_DISK       9
ACTION_WRITE             'W'
ACTION_READ              'R'
ACTION_FREE_LOCK         15
ACTION_DELETE_OBJECT     16
ACTION_RENAME_OBJECT     17
ACTION_COPY_DIR          19
ACTION_WAIT_CHAR         20
ACTION_SET_PROTECT       21
ACTION_CREATE_DIR        22
ACTION_EXAMINE_OBJECT    23
ACTION_EXAMINE_NEXT      24
ACTION_DISK_INFO         25
ACTION_INFO              26
```

```
ACTION_SET_COMMENT      28
ACTION_PARENT           29
ACTION_TIMER            30
ACTION_INHIBIT          31
ACTION_DISK_TYPE        32
ACTION_DISK_CHANGE      33
```

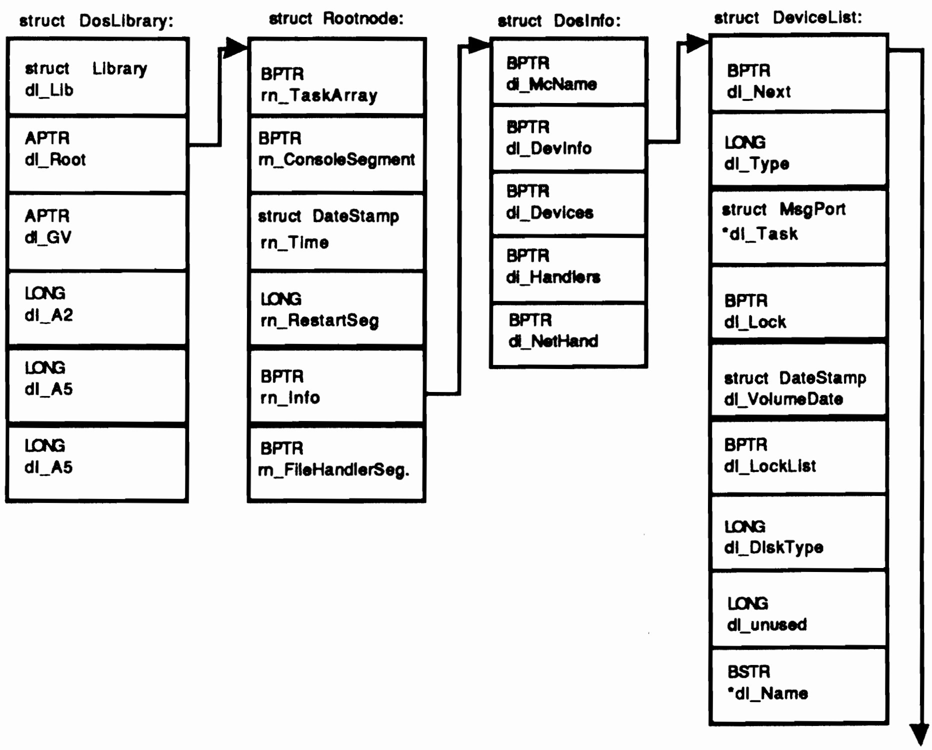
6.2.7 DosBase

Structures:

```
struct DosLibrary <libraries/dosextens.h>
{
0x00  0  struct Library dl_lib;
0x22 34  APTR   dl_Root;          /* pointer to the RootNode */
0x26 38  APTR   dl_GV;
0x2A 42  LONG   dl_A2;
0x2E 46  LONG   dl_A5;
0x32 50  LONG   dl_A6;
0x36 54
};
struct RootNode <libraries/dosextens.h>
{
0x00  0  BPTR   rn_TaskArray; /* [0] = Maximum number of CLIs */
                              /* [n] APTR to process from CLI n */
0x04  4  BPTR   rn_ConsoleSegment;
0x08  8  struct DateStamp rn_Time;            /* Current time */
0x14 20  LONG   rn_RestartSeg;
0x18 24  BPTR   rn_Info;                /* pointer to DosInfo */
0x1C 28  BPTR   rn_FileHandlerSegment;
0x20 32
};
struct DosInfo <libraries/dosextens.h>
{
0x00  0  BPTR  di_McName;
0x04  4  BPTR  di_DevInfo;  /* pointer to list of all devices */
0x08  8  BPTR  di_Devices;
0x0C 12  BPTR  di_Handlers;
0x10 16  APTR  di_NetHand;
0x14 20
};
struct DeviceList <libraries/dosextens.h>
{
0x00  0  BPTR             dl_Next;     /* BPTR to next device */
0x04  4  LONG             dl_Type;
0x08  8  struct MsgPort  *dl_Task;
0x0C 12  BPTR             dl_Lock;        /* Not for disks    */
0x10 16  struct DateStamp dl_VolumeDate;
0x1C 28  BPTR             dl_LockList;  /* List of all locks */
0x20 32  LONG             dl_DiskType;  /* e.g., "DOS"       */
0x24 36  LONG             dl_unused;
0x28 40  BSTR            *dl_Name;    /* BPTR to BCPL string */
0x2C 44
};

Device Type (dl_Type):
DLT_DEVICE     0
DLT_DIRECTORY  1
DLT_VOLUME     2
```

6.3 The Intuition library

The Intuition library is resident, which means that it stays in memory right after the computer is booted. Intuition handles everything concerning the graphic user interface, from screen and window management to the report transfer between gadgets and programs. The `OpenLibrary` function allows us access to all the programs ("intuition.library" 0L). This library has a wealth of functions for controlling the windows, gadgets, menus, graphics and screens.



The Intuition library operates in conjunction with the Graphics and the Layers libraries. Intuition uses the Layers library to represent the gadgets and window overlapping. It also uses the Graphics library for graphic output and formatting, fonts, the bit-map, ViewPort management and everything else that has a task.

Intuition library functions

6.3.1 Window functions

6.3.2 Gadget functions

6.3.3 Menu functions

6.3.4 Requester and alert functions

6.3.1. Window functions

ActivateWindow **Activates Intuition window**

Syntax:

```
ActivateWindow(Window);
     -450        A0
struct Window *Window;
```

Description: This function activates a window controlled by Intuition.

Menu or gadget access can slow execution speed. When the window is really active, then it can easily be checked using the `ACTIVEWINDOW` `IDCMP` flags.

The function should be used with caution because keyboard input gives you a new active window.

Parameter: Window: Pointer to an Intuition window structure.

Warning: Intuition report processing can be paralyzed if the function is called frequently.

See Also: `OpenWindow()`, IDCMP flags (`ACTIVEWINDOW`)

CloseWindow **Closes an Intuition window**

Syntax:

```
CloseWindow(Window);
    -72        A0
struct Window *Window;
```

Description: This function closes a window managed from Intuition, removes the window from the system list and releases allocated memory, closing the system screen if needed. The system ignores any reports in the IDCMP.

When a message port to the window is opened, you should ensure that there are no unprocessed reports there. The access to a memory buffer that no longer exists can cause a system error.

You should also know that a previously added menu strip must be removed before you may close a window. This Intuition function blocks all other functions until it has ended the task.

Parameter: Window: Pointer to an Intuition window structure.

See also: `OpenWindow(),CloseScreen()`

ModifyIDCMP — Changes IDCMP flag settings

Syntax:

```
ModifyIDCMP(Window, IDCMPFlags);
    -150     A0        D0
struct Window *Window;
ULONG IDCMPFlags;
```

Description: This function changes the IDCMP (Intuition Direct Communication Message Port) flags of an Intuition window and places it in a new status. A port may be added where one did not previously exist by setting unset flags; some flags may be unset by setting others; and a port may be removed by entering a zero.

Parameters:

Window: Pointer to an Intuition window structure.
IDMCPFlags: Flag bits describing the status of the IDCMP.

See Also: `OpenWindow(),CloseWindow()`

MoveWindow — Moves window to another position

Syntax:

```
MoveWindow(Window, DeltaX, DeltaY);
   -168      A0      D0      D1
struct Window *Window;
SHORT DeltaX, DeltaY;
```

Description: This function moves an Intuition window in the `DeltaX` and `DeltaY` directions. The new positioning does not execute immediately. When Intuition receives an input event (which it does at a rate of 10 times per second), the function executes.

Parameters:

Window: Pointer to an Intuition window structure.
DeltaX: Integer value specifying movement in the X direction.
DeltaY: Integer value specifying movement in the Y direction.

Warning: This functions does not check the movement values for accuracy. Make sure that your coding doesn't try moving the window to a nonexistent location, or a system error may result.

See Also: `SizeWindow()`

OpenWindow — Opens an Intuition window

Syntax:

```
Window = OpenWindow(NewWindow);
  D0       -204       A0
struct NewWindow NewWindow;
```

Description: This function opens an Intuition window with the values specified in the structure. A `window` structure is added to the window. The `NewWindow` structure is no longer needed and can be removed.

If the window appears on a `CUSTOMSCREEN`, this screen can be opened before opening the window. If the window should include gadgets, a linked list of these must be included in the `NewWindow` structure.

Parameter:

NewWindow: Pointer to a `NewWindow` structure that was installed beforehand by the programmer.

Result:

Window: Pointer to an Intuition window structure.

See Also: `CloseWindow(),ModifyIDCMP(),OpenScreen(), CloseScreen(),WindowTitles()`

Structures:

```
struct NewWindow <intuition/intuition.h>
{
0x00  00  SHORT LeftEdge; /* Upper left corner of window
                             relative to the screen */
0x02  02  SHORT TopEdge;
0x04  04  SHORT Width;   /* Window width and height in pixels */
0x06  06  SHORT Height;
0x08  08  UBYTE DetailPen; /* Color number of foreground pen */
0x09  09  UBYTE BlockPen; /* Color number of the background
                             pen */
0x0A  10  ULONG IDCMPFlags;     /* All of the set IDCMP flags */
0x0E  14  ULONG Flags;         /* All of the set window flags */
0x12  18  struct Gadget *FirstGadget; /* Pointer to the first
                                         gadget */
0x16  22  struct Image *CheckMark;  /* Pointer to the graphic
                                       from the checkmark */
0x1A  26  UBYTE *Title;   /* Pointer to the title text string */
0x1E  30  struct Screen *Screen;     /* Pointer to the screen */
0x22  34  struct Bitmap *Bitmap;        /* Pointer to a bitmap
                                           for the window */
0x26  38  SHORT MinWidth;              /* Minimum window size */
0x28  40  SHORT MinHeight;
0x2A  42  USHORT MaxWidth;             /* Maximum window size */
0x2C  44  USHORT MaxHeight;
0x2E  46  USHORT Type;                        /* Window type */
0x30  48
};
```

```
struct Window <intuition/intuition.h>
{
0x00  00   struct Window *NextWindow; /* Pointer to other windows
                                         in the screen */
0x04  04   SHORT LeftEdge; /* Top left corner of window relative
                              to the screen */
0x06  06   SHORT TopEdge;
0x08  08   SHORT Width;   /* Width and height of the window in
                             pixels */
0x0A  10   SHORT Height;
0x0C  12   SHORT MouseY;     /* Mouse position relative to the
                                window */
0x0E  14   SHORT MouseX;
0x10  16   SHORT MinWidth;             /* Minimum window size */
0x12  18   SHORT MinHeight;
0x14  20   USHORT MaxWidth;            /* Maximum window size */
0x16  22   USHORT MaXHeight;
0x18  24   ULONG Flags;                       /* Window flags */
0x1C  28   struct Menu *MenuStrip;     /* Pointer to the menu
                                          list of this Window */
0x20  32   UBYTE *Title;   /* Pointer to the title text of the
                              window */
0x24  36   struct Requester *FirstRequest;   /* Pointer to the
                                                first requester
                                                in the window  */
0x28  40   struct Requester *DMRequest;    /* Pointer to the
                                              DoubleMenuRequester
                                              of the window */
0x2C  44   SHORT ReqCount;    /* Number of opened requester in
                                 window */
0x2E  46   struct Screen *WScreen;     /* Pointer to WB screen */
0x32  50   struct RastPort *RPort;   /* Pointer to Rastport of
                                        window */
0x36  54   BYTE BorderLeft;                   /* Margin width */
0x37  55   BYTE BorderTop;
0x38  56   BYTE BorderRight;
0x39  57   BYTE BorderBottom;
0x3A  58   struct RastPort *BorderRPort;  /* Pointer to RastPort
                                             of the window margin */
0x3E  62   struct Gadget *FirstGadget;   /* Pointer to the first
                                            gadget in the window */
0x42  66   struct Window *Parent     /* Pointer to the previous
                                        window */
0x46  70   struct Window *Descendant;   /* Pointer to window to
                                           close */
0x4A  74   USHORT *Pointer;    /* Pointer to the mouse pointer
                                  graphic */
0x4E  78   BYTE PtrHeight;            /* Mouse pointer height */
0x4F  79   BYTE PtrWidth;               /* Mouse pointer width*/
0x50  80   BYTE XOffset;                   /* Marking HotSpot */
0x51  81   BYTE YOffset;
0x52  82   ULONG IDCMPFlags;               /* All IDCMP flags */
0x56  86   struct MsgPort *UserPort;  /* Message Port for user */
0x5A  90   struct MsgPort *WindowPort;     /* Message Port for
                                              window */
0x5E  94   struct IntuiMessage *MessageKey;  /* Report from
                                                Intuition */
0x62  98   UBYTE DetailPen;      /* Foreground character color */
0x63  99   UBYTE BlockPen;       /* Background character color */
0x64 100   struct Image *CheckMark; /* Pointer to graphic for
                                       checkmark */
0x68 104   UBYTE *ScreenTitle;    /* Pointer to the screen title
```

```
                                            text */
0x6C 108  SHORT GZZMouseX;            /* Mouse position in the GZZ
                                            Window */
0x6E 110  SHORT GZZMouseY;
0x70 112  SHORT GZZWidth;           /* GZZ window width and height */
0x72 114  SHORT GZZHeight;
0x74 116  UBYTE *ExtData;            /* Pointer to more data
                                           (Expansions) */
0x78 120  BYTE *UserData;           /* Pointer to window user data */
0x7C 124  struct Layer *WLayer;     /* Pointer to window layer */
0x80 128  struct TextFont *IFont; /* Pointer to standard
                                             character set in this
                                             window */
0x84 132
};

Window_Flags:
WINDOWSIZING         0x0001L /* Window gadgets */
WINDOWDRAG           0x0002L
WINDOWDEPTH          0x0004L
WINDOWCLOSE          0x0008L
SIZEBRIGHT           0x0010L
SIZEBBOTTOM          0x0020L
REFRESHBITS          0x00C0L /* Refresh modes */
SMART_REFRESH        0x0000L
SIMPLE_REFRESH       0x0040L
SUPER_BITMAP         0x0080L
OTHER_REFRESH        0x00C0L
BACKDROP             0x0100L
REPORTMOUSE          0x0200L
GIMMEZEROZERO        0x0400L
BORDERLESS           0x0800L
ACTIVATE             0x1000L
WINDOWACTIVE         0x2000L
INREQUEST            0x4000L
MENUSTATE            0x8000L
RMBTRAP              0x00010000L
NOCAREREFRESH        0x00020000L
WINDOWREFRESH        0x01000000L
WBENCHWINDOW         0x02000000L
WINDOWTICKED         0x04000000L
SUPER_UNUSED         0xFCFC0000L
```

RefreshWindowFrame — Redraws window

Syntax:

```
RefreshWindowFrame(Window);
       -456        A0
struct Window *Window;
```

Description: This function redraws the border, title and gadgets of a window, keeping graphic distortion to a minimum.

Parameter: Window: Pointer to an Intuition window structure.

See Also: RefreshGadgets(), RefreshGList()

SetWindowTitles — Sets window/screen titles

Syntax:

```
SetWindowTitles(Window, WindowTitle, ScreenTitle);
      -276       A0           A1          A2
struct Window *Window
UBYTE *WindowTitle, *ScreenTitle
```

Description: This function specifies the screen or window title text.

The window title is announced from Intuition. The window title only appears in the screen title list if the window is also active. After the call of the routine the titles of the screen and window change. If you only want to change one of the two titles, you replace the other pointer with a -1. This informs Intuition that the previous text should be retained.

Parameters:

Window: Pointer to an Intuition window structure.
WindowTitle: Pointer to a string that ends with zero that represents the text. This string can also contain a value of -1 for original text or a value of 0 for no text.
ScreenTitle: Pointer to a string representing the text, that ends with zero. This string can also contain a value of -1 for the original text or a value of 0 for no text.

See Also: `OpenWindow(),RefreshWindowFrame(),OpenScreen()`

SizeWindow — Changes window size

Syntax:

```
SizeWindow(Window, DeltaX, DeltaY);
    -288     A0      D0      D1
struct Window *Window;
SHORT DeltaX, DeltaY;
```

Description: This function changes an Intuition window's size in the `DeltaX` and `DeltaY` directions. The new size does not execute immediately. When Intuition receives an input event (which it does at a rate of 10 times per second), the function executes.

Parameters:

Window: Pointer to an Intuition window structure.
DeltaX: Integer value which specifies resizing in the X direction.
DeltaY: Integer value which specifies resizing in the Y direction.

Warning: This function does not check the sizing values for accuracy. Make sure that your coding doesn't try resizing the window in a nonexistent location, or a system error may result.

See Also: `MoveWindow(),OpenWindow()*`

WindowLimits — Window's minimum/maximum values

Syntax:

```
Settings = WindowLimits(Window, MinWidth, MinHeight, MaxWidth,
    D0          -318       A0       D0         D1          D2
                                                    MaxHeight);
                                                        D3
BOOL Settings;
struct Window *Window;
USHORT MinWidth, MinHeight;
USHORT MaxWidth, MaxHeight;
```

Description: This function determines the minimum and maximum sizes of the window. After the call the window size can be changed to match these values.

If you don't want to change these values, set them to zero. The function then ignores further entries and maintains the original settings. Entering -1 changes the window to maximum (screen) size.

Parameters:

Window:	Pointer to an Intuition window structure.
MinWidth:	New minimum window width.
MinHeight:	New minimum window height.
MaxWidth:	New maximum window width.
MaxHeight:	New maximum window height.

Result: Returns TRUE if all parameters are within allowable limits. If a value does not lie in the region (too large or too small), FALSE is returned to you.

Comments: When the function is called during resizing of the window, the new values take effect after this operation.

See Also: `GetScreenData()`

WindowToBack — Moves window to back

Syntax:

```
WindowToBack(Window);
    -306      A0
struct Window *Window;
```

Description: This function moves the specified window behind all others on the screen.

The function does not execute immediately. When Intuition receives an input event (input events occur 10 times per second), the function executes.

Parameter: Window: Pointer to an Intuition window structure.

Comments: The function does not operate with BACKDROP windows.

See Also: `WindowToFront(),MoveWindow(),SizeWindow()`

WindowToFront **Moves window to front**

Syntax:

```
WindowToFront(Window);
     -312       A0
struct Window *Window;
```

Description: This function moves the specified window in front of all others on the screen.

The function does not execute immediately. When Intuition receives an input event (input events occur 10 times per second), the function executes.

Parameter: Window: Pointer to an Intuition window structure.

Comments: The function does not operate with BACKDROP windows.

See Also: `WindowToBack(),MoveWindow(),SizeWindow()`

6.3.2 Gadget functions

ActivateGadget **Activates string gadget**

Syntax:

```
Succes = ActivateGadget(Gadget, Window, Requester);
  D0            -462       A0      A1        A2
BOOL Succes;
struct Gadget *Gadget;
struct Window *Window;
struct Requester *Request;
```

Description: This function activates a string gadget, provided the `REQGADGET` flag in the gadget structure is set.

Parameters:

Gadget:	Pointer to the string gadget to be activated.
Window:	Pointer to the window structure linked in the gadget.
Requester:	Pointer to a requester when the string gadget is found in such.

Result: Returns TRUE if all conditions are fulfilled (see Comments below), or FALSE if conditions are unfulfilled.

Comments: The window containing the gadget must be active before execution.

No other gadgets may be in use during the function call (including system gadgets).

If this function involves a requester, the requester must also be active.

The function does not operate during a menu choice.

AddGadget — Adds gadget to window's gadget list

Syntax:

```
RealPosition = AddGadget(Window, Gadget, Position);
     D0             -42       A0      A1       D0
USHORT RealPosition;
struct Window *Window;
struct Gadget *Gadget;
USHORT Position;
```

Description: This function adds the specified gadget to the active window's list of gadgets.

If you use system gadgets, these are placed at the beginning of the gadget list so that they are always checked first.

Intuition does not support user-inserted screen gadgets. Instead you can open a BACKDROP window, which looks similar to a screen, and add gadgets to the BACKDROP window.

Parameters:

Window:	Pointer to an Intuition window structure.
Gadget:	Pointer to a gadget structure.
Position:	Gadget's position number: `Position` = 0: First gadget in list. `Position` = 1: Second gadget in list. `Position` = 2: Third gadget, etc. `Position` = -1: Last gadget in list.

Result: Returns the position at which the gadget is inserted.

See Also: `AddGList(),RemoveGadget(),RemoveGList()`

Structures:

```
struct Gadget <intuition/intuition.h>
{
0x00  00  struct Gadget *NextGadget; /* Link to next gadget in
                                          list */
0x04  04  SHORT LeftEdge; /* Pixel position of the click region:
                             upper left corner */
0x06  06  SHORT TopEdge;
0x08  08  SHORT Width; /* Width and height of click region in
                          pixels */
0x0A  10  SHORT Height;
0x0C  12  USHORT Flags;      /* Gadget flags */
0x0E  14  USHORT Activation; /* Activation mode */
```

```
0x10  16  USHORT GadgetType; /* Flags for gadget types */
0x12  18  APTR GadgetRender; /* Pointer to structure of
                                gadget's appearance */
0x16  22  APTR SelectRender; /* Pointer to graphic
                                activation */
0x1A  26  struct IntuiText *GadgetText; /* Pointer to gadget's
                                           text */
0x1E  30  LONG MutualExclude;/* Deactivation of other gadgets
                                (not currently implemented) */
0x22  34  APTR SpecialInfo;  /* Pointer to additional
                                information */
0x26  38  USHORT GadgetID;   /* Gadget identification number */
0x28  40  APTR UserData;     /* Pointer to program data */
0x2C  44
};
```

Gadget_Flags:

```
GADGHIGHBITS 0x0003L/* Gadget highlighting flags */
GADGHCOMP 0x0000L   /* Click region inverted */
GADGHBOX 0x0001L    /* Draws box around the click region */
GADGHIMAGE 0x0002L  /* New graphic displayed */
GADGHNONE 0x0003L   /* No reaction from gadget */
GADGIMAGE 0x0004L   /* Graphic displayed instead of a margin */
GRELBOTTOM 0x0008L  /* Position relative to bottom
                       window border */
GRELRIGHT 0x0010L   /* Position relative to right border */
GRELWIDTH 0x0020L   /* Relative width */
GRELHEIGHT 0x0040L  /* Relative height */
SELECTED 0x0080L    /* Gadget selected */
GADGDISABLED 0x0100L/* Gadget cannot be selected */
```

Gadget_Activation:

```
RELVERIFY 0x0001L      /* Verifies selection */
GADGIMMEDIATE 0x0002L /* Selection executes on 1 click */
ENDGADGET 0x0004L     /* Requester gadget that goes to the end */
FOLLOWMOUSE 0x0008L    /* Follows mouse coordinates */
RIGHTBORDER 0x0010L    /* Gadget placed in right window border */
LEFTBORDER 0x0020L     /* Gadget placed in left window border */
TOPBORDER 0x0040L      /* Gadget placed in top window border */
BOTTOMBORDER 0x0080L /* Gadget placed in bottom window border */
TOGGLESELECT 0x0100L  /* Toggles gadget on/off */
STRINGCENTER 0x0200L /* Shoe text in middle of string gadget */
STRINGRIGHT 0x0400L    /* Displays text in right margin */
LONGINT 0x0800L        /* Long word integer gadget */
ALTKEYMAP 0x1000L      /* Gadget uses alternate keymap */
```

Gadget_GadgetType:

```
BOOLEXTEND 0x2000L
GADGETTYPE 0xFC00L     /* Gadget types */
SYSGADGET 0x8000L      /* System gadget */
SCRGADGET 0x4000L      /* Screen gadget */
GZZGADGET 0x2000L      /* GimmeZeroZero gadget */
REQGADGET 0x1000L      /* Requester gadget */
SIZING 0x0010L         /* Sizing gadget */
WDRAGGING 0x0020L      /* Window Drag gadget */
SDRAGGING 0x0030L      /* Screen Drag gadget */
WUPFRONT 0x0040L       /* Window Up Front gadget */
```

```
SUPFRONT 0x0050L       /* Screen Up Front gadget */
WDOWNBACK 0x0060L      /* Window Down Back gadget */
SDOWNBACK 0x0070L      /* Screen Down Back gadget */
CLOSE 0x0080L          /* Close gadget */
BOOLGADGET 0x0001L     /* Boolean gadget (in/out) */
GADGET0002 0x0002L     /* Type 0002 gadget */
PROPGADGET 0x0003L     /* Proportional gadget */
STRGADGET 0x0004L      /* String gadget */
struct BoolInfo <intuition/intuition.h>
{
0x00  00  USHORT Flags; /* addition structure for Boolean
                           gadgets */
0x02  02  UWORD *Mask;
0x04  04  ULONG Reserved;
0x08  08
};
```

BoolInfo_Flags:

```
BOOLMASK 0x0001L /* User-defined Boolean mask */
struct PropInfo
{
0x00  00  USHORT Flags;      /* Proportional gadget flags*/
0x02  02  USHORT HorizPot;   /* Horizontal position of slider */
0x04  04  USHORT VertPot;    /* Vertical position */
0x06  06  USHORT HorizBody;  /* Horizontal size of slider */
0x08  08  USHORT VertBody;   /* Vertical size */
0x0A  10  USHORT CWidth;     /* Container width*/
0x0C  12  USHORT CHeight;    /* Container height */
0x0E  14  USHORT HPotRes;   /* Horizontal resolution of slider */
0x10  16  USHORT VPotRes;    /* Vertical resolution */
0x12  18  USHORT LeftBorder;/* Left border of proportional
                                gadget */
0x14  20  USHORT TopBorder;  /* Top border */
0x16  22
};
```

PropInfo_Flags:

```
AUTOKNOB 0x0001L  /* Default slider knob */
FREEHORIZ 0x0002L /* Slider can be moved horizontally */
FREEVERT 0x0004L  /* Slider can be moved vertically */
PROPBORDERLESS 0x0008L /* No box around slider */
KNOBHIT 0x0100L        /* Set when slider is active */
```

PropInfo_Values:

```
KNOBHMIN 6L     /* Minimum horizontal knob resolution */
KNOBVMIN 4L     /* Minimum vertical resolution */
MAXBODY 0xFFFFL /* Maximum */
MAXPOT 0xFFFFL
struct StringInfo <intuition/intuition.h>
{
0x00  00  UBYTE *Buffer;      /* Pointer to text buffer */
0x04  04  UBYTE *UndoBuffer;  /* Pointer to undo buffer */
0x08  08  SHORT BufferPos;    /* Cursor position in buffer */
0x0A  10  SHORT MaxChars;     /* Maximum number of characters in
                                 buffer */
0x0C  12  SHORT DispPos;      /* Buffer position in display */
0x0E  14  SHORT UndoPos;     /* Cursor position in undo buffer */
```

```
0x10  16  SHORT NumChars;    /* Number of characters entered */
0x12  18  SHORT DispCount; /* Number of characters in display */
0x14  20  SHORT CLeft;       /* Left border of container */
0x16  22  SHORT CTop;        /* Top border */
0x18  24  struct Layer *LayerPtr; /* Pointer to gadget layer */
0x1C  28  LONG LongInt; /* Number re-converted for integer
                                 gadget */
0x20  32  struct KeyMap *AltKeyMap; /* Pointer to an alternate
                                    keymap */
0x24  36
};
```

AddGList — Inserts gadget list

Syntax:

```
RealPosition = AddGList(Window, Gadget,
    D0              -438     A0      A1
          Position, Numgad, Requester);
             D0       D1        D2
USHORT RealPosition;
struct Window *Window;
struct Gadget *Gadget;
USHORT Position;
USHORT Numgad;
struct Requester *Requester;
```

Description: This function inserts a list of gadgets in an existing window gadget list or requester gadget list. The `*Requester` pointer comes into play only when gadgets must be added to a requester contained in a given window.

The `AddGList` function adds as many gadgets of the linked list to that of the window as are given with Numgad. It is interrupted when a pointer to the next gadget is set to zero.

Parameters:

Window:	Pointer to window structure in which the new gadget should be inserted.
Gadget:	Pointer to gadget structure.
Position:	Gadget's position number.
Numgad:	Number of the gadget to be added to the list.
Requester:	Pointer to gadget's request structure.

Result: Returns the position at which the gadget was actually inserted. This number is in effect until no other gadgets are inserted.

Warning: If you don't use all of the gadgets of a linked list for the window or the requester, make sure that Intuition changes the pointer of the last gadget inserted.

See Also: `AddGadget()`, `RemoveGadget()`, `RemoveGList()`

ModifyProp — Changes proportional gadget settings

Syntax:

```
ModifyProp(Gadget, Window, Requester, Flags,
    -156       A0      A1         A2       D0
    HorizPot, VertPot, HorizBody, VertBody);
        D1       D2        D3         D4
struct Gadget *Gadget;
struct Window *Window;
struct Requester *Requester;
USHORT Flags;
USHORT HorizPot, VertPot;
USHORT HorizBody, VertBody;
```

Description: This function changes the settings assigned to a proportional gadget, then displays the new gadget. The normal `Refresh` function redisplays the other gadgets on the list as well as the proportional gadget. The settings are similar to those used for a requester.

Parameters:

Gadget:	Pointer to proportional gadget structure.
Window:	Pointer to window structure containing the gadget.
Requester:	Pointer to requester structure containing the gadget.
Flags:	Value that should be transferred to the `Flags` variable of the gadget.
HorizPot:	Value that should be transferred to the `HorizPot` variable of the gadget.
VertPot:	Value that should be transferred to the `VertPot` variable of the gadget.
HorizBody:	Value that should be transferred to the `HorizBody` variable of the gadget.
VertBody:	Value that should be transferred to the `VertBody` variable of the gadget.

See Also: `NewModifyProp()`

NewModifyProp — Changes proportional gadget settings

Syntax:

```
NewModifyProp(Gadget, Window, Requester, Flags, HorizPot,
      -468        A0      A1         A2       D0      D1
                  VertPot, HorizBody, VertBody, Numgad);
                     D2        D3         D4       D5
struct Gadget *Gadget;
struct Window *Window;
struct Requester *Requester;
USHORT Flags;
USHORT HorizPot, VertPot;
USHORT HorizBody, VertBody;
int Numgad
```

Description: The command has the same function as the `ModifyProp()` command. How many gadgets should be re-drawn can also be set through `Numgad`.

Description: This function changes the settings assigned to a proportional gadget, then displays the new gadget, refreshing only the gadgets specified by the `Numgad` parameter.

Parameters:

- Gadget: Pointer to proportional gadget structure.
- Window: Pointer to window structure containing the gadget.
- Requester: Pointer to requester structure containing the gadget.
- Flags: Value that should be transferred to the `Flags` variable of the gadget.
- HorizPot: Value that should be transferred to the `HorizPot` variable of the gadget.
- VertPot: Value that should be transferred to the `VertPot` variable of the gadget.
- HorizBody: Value that should be transferred to the `HorizBody` variable of the gadget.
- VertBody: Value that should be transferred to the `VertBody` variable of the gadget.
- Numgad: Number of gadget that should be re-drawn. Entering -1 is the same as accessing `ModifyProp`.

See Also: `ModifyProp()`

OffGadget — Disables gadget

Syntax:

```
OffGadget(Gadget, Window, Requester);
   -174     A0      A1       A2
struct Gadget *Gadget;
struct Window *Window;
struct Requester *Requester;
```

Description: This function makes it impossible for the user to choose a gadget. A disabled gadget appears in ghost print. The `GADGDISABLED` flag can also be set through programming to get the same result.

Parameters:

- Gadget: Pointer to gadget structure.
- Window: Pointer to window structure containing the gadget.
- Requester: Pointer to requester structure containing the gadget.

See Also: `OnGadget()`

OnGadget — Enables gadget

Syntax:

```
OnGadget(Gadget, Window, Requester);
  -186     A0      A1       A2
struct Gadget *Gadget;
struct Window *Window;
struct Requester *Requester;
```

Description: This function makes it possible for the user to choose a previously disabled gadget. A disabled gadget appears in ghost print. The `GADGDISABLED` flag can also be unset through programming to get the same result. The `OnGadget` function executes a refresh that applies to this gadget and any gadget following in the window list.

Parameters:

Gadget: Pointer to gadget structure.
Window: Pointer to window structure containing the gadget.
Requester: Pointer to requester structure containing the gadget.

See Also: `OffGadget()`

RefreshGadgets — Redraws gadgets

Syntax:

```
RefreshGadgets(Gadgets, Window, Requester);
     -222          A0      A1        A2
struct Gadget *Gadget;
struct Window *Window;
struct Requester *Requester;
```

Description: This function redraws all of the gadgets in a window list, beginning with the specified gadget. A window graphic may be disturbed by drawing a new graphic, and require a gadget refresh. For example, if you want to change a gadget's settings, you must remove the gadget from the window with `RemoveGadget()` or `RemoveGList()`, change your settings and re-insert the gadget in the list using `AddGadget()` or `AddGList()`. The gadget may not always be represented again. Invoking `RefreshGadgets()` allows Intuition to redisplay the gadget graphic.

Parameters:

Gadget: Pointer to gadget structure.
Window: Pointer to window structure containing the gadget.
Requester: Pointer to requester structure containing the gadget.

See Also: `RefreshGList()`, `RemoveGadget()`, `RemoveGList()`, `AddGadget()`, `AddGList()`

RefreshGList — Redraws gadget list

Syntax:

```
RefreshGList(Gadgets, Window, Requester, Numgad);
    -432        A0       A1        A2        D0
struct Gadget *Gadget;
struct Window *Window;
struct Requester *Requester;
SHORT Numgad;
```

Description: This function redraws a certain number of gadgets in a window list, beginning with the specified gadget. The `RefreshGList` function operates in a manner similar to the `RefreshGadgets` function.

Parameters:

Gadget: Pointer to gadget structure.
Window: Pointer to window structure containing the gadget.
Requester: Pointer to requester structure containing the gadget.
Numgad: Number of gadgets that should be redrawn. Entering a -1 for this parameter has the same effect as selecting `RefreshGadgets()`. Entering a -2 for this parameter redraws all of the gadgets in the requester list (applies to requester gadgets only).

See Also: `RefreshGadgets()`, `RemoveGadget()`, `RemoveGList()`, `AddGadget()`, `AddGList()`

RemoveGadget — Removes gadget from window list

Syntax:

```
Position = RemoveGadget(Window, Gadget);
    D0          -228       A0      A1
USHORT Position;
struct Window *Window;
struct Gadget *Gadget;
```

Description: This function removes the specified gadget from the window list Gadgets that are present in a window's requester.

Parameters:

Window: Pointer to the window containing the gadget or requester.
Gadget: Pointer to the gadget structure that should be removed (user must specify whether the gadget is part of a window or requester).

Result: Returns either the gadget's previous position or a value of -1. The -1 can mean that the gadget was not present in the list, the list has no gadgets at all or that the 65535th gadget was removed.

See Also: `AddGadget()`, `AddGList()`, `RemoveGList()`

RemoveGList — Removes gadgets from window list

Syntax:

```
Position = RemoveGList(Window, Gadget, Numgad);
    D0         -444       A0      A1      D0
USHORT Position;
struct Window *Window;
struct Gadget *Gadget;
SHORT Numgad
```

Description: This function removes the specified number of gadgets, beginning with the one defined by `Gadget`. It defaults to the first gadget if the gadgets are in a requester. See also `RemoveGadget()`.

Parameters:

Window: Pointer to window structure containing the gadget or requester to be removed.
Gadget: Pointer to the first gadget to be removed.
Numgad: Number of gadgets to be removed. Entering a -1 for this parameter removes all gadgets up to the end of the gadget list.

Result: Returns either the gadget's previous position or a value of -1. The -1 can mean that the gadget was not present in the list, the list has no gadgets at all or that the 65535th gadget was removed.

See Also: `RemoveGadget(),AddGadget(),AddGList()`

6.3.3 Menu functions

ClearMenuStrip — Removes menu strip from window

Syntax:

```
ClearMenuStrip(Window);
      -54         A0
struct Window *Window;
```

Description: This function removes the menu strip from the window. If menus are currently being accessed, the `ClearMenuStrip` function executes after the menu access ends.

Parameter:

Window: Pointer to window containing the menu strip.

Comments: This function must be called before making any changes to the structure.

See Also: `SetMenuStrip()`

ItemAddress — Returns menu item address

Syntax:

```
ItemAddress = ItemAddress(MenuStrip, MenuNumber);
     D0           -144          A0          D0
struct MenuItem *ItemAddress;
struct Menu *MenuStrip;
USHORT MenuNumber;
```

Description: This function provides the pointer to a menu item's corresponding structure. This pointer is required when you want to change a menu item's settings.

Parameters:

MenuStrip: Pointer to menu strip.
MenuNumber: Number of the (sub)menu item.

Result: ItemAddress: Pointer to the `MenuItem` structure selected by `MenuNumber`.

OffMenu — Disables menu or menu item

Syntax:

```
OffMenu(Window, MenuNumber);
 -180    A0       D0
struct Window *Window;
USHORT MenuNumber;
```

Description: This function disables a menu item or entire menu. Submenu items in a menu disabled using `OffMenu` are also inaccessible.

Parameters:
Window: Pointer to window containing the menu strip.
MenuNumber: Menu/item that should be displayed in ghost print.

See Also: `OnMenu()`

OnMenu — Enables menu or menu item

Syntax:

```
OnMenu(Window, MenuNumber);
 -192   A0       D0
struct Window *Window;
USHORT MenuNumber;
```

Description: This function enables a menu item or entire menu. Submenu items in a menu enabled using `OnMenu` are also accessible.

Parameters:
Window: Pointer to window containing the menu strip.
MenuNumber: Menu/item that should be enabled.

See Also: `OffMenu()`

SetMenuStrip — removes menu strip from window

Syntax:

```
Success = SetMenuStrip(Window, Menu);
   D0          -264      A0     A1
struct Window *Window;
struct Menu *Menu;
```

Description: This function adds the predefined menu strip to the window.

Parameters:
Window: Pointer to window into which the menu strip should be inserted.
Menu: Pointer to the linked list from the menu strip.

Result: Success: Returns TRUE if no error occurs. This result is consistent, because the function waits until everything executes without error.

Warning: When using a menu, make sure that a menu strip is removed from the window before closing the window, or an error may occur.

See Also: `ClearMenuStrip()`

Structures:

```
struct Menu <intuition/intuition.h>
{
0x00  00  struct Menu *NextMenu; /* Link to next menu
                                    structure */
0x04  04  SHORT LeftEdge;      /* Pixel position of upper left
                                  corner */
0x06  06  SHORT TopEdge;
0x08  08  SHORT Width;         /* Menu text width and height */
0x0A  10  SHORT Height;
0x0C  12  USHORT Flags;         /* Menu attribute flags */
0x0E  14  BYTE *MenuName;       /* Pointer to menu string */
0x0F  18  struct MenuItem *FirstItem; /* Pointer to first menu
                                         item */
0x16  22  SHORT JazzX; /* Internal management values */
0x18  24  SHORT JazzY;
0x10  26  SHORT BeatX;
0x12  28  SHORT BeatY;
0x14B 30
};
```

Menu_Flags:

```
MENUENABLED 0x0001L /* Menu item enabled */
MIDRAWN 0x0100L     /* Menu item drawn */
struct MenuItem <intuition/intuition.h>
{
0x00  00  struct MenuItem *NextItem; /* Link to next menu item
                                        structure */
0x04  04  SHORT LeftEdge       /* Pixel position of menu item's
                                  upper left corner*/
0x06  06  SHORT TopEdge;
0x08  08  SHORT Width;              /* Width in pixels */
0x0A  10  SHORT Height;         /* Height in pixels */
0x0C  12  USHORT Flags;      /* Menu attribute flags */
0x0E  14  LONG MutualExclude; /* Exclude certain menu
                                 items from activation */
0x12  18  APTR ItemFill;   /* Pointer to normal display */
0x16  22  APTR SelectFill; /* Pointer to selected display */
0x1A  26  BYTE Command;    /* Keyboard character instead of menu
                              item */
0x1B  27  struct MenuItem *SubItem;  /* pointer to the first
                                        submenu item in menu */
0x1F  31  USHORT NextSelect;        /* Next menu number for
                                       multiple select*/
0x21  33
};
```

MenuItem_Flags:

```
CHECKIT 0x0001L    /* Menu item checked */
ITEMTEXT 0x0002L   /* Text display only (no graphics) */
COMMSEQ 0x0004L    /* One character for kbd shortcut */
MENUTOGGLE 0x0008L /* Toggled menu item */
ITEMENABLED 0x0010L/* Menu item enabled */
```

```
HIGHFLAGS 0x00C0L  /* All highlighting flags */
HIGHIMAGE 0x0000L  /* Graphic display */
HIGHCOMP 0x0040L   /* Inverse menu item when selected */
HIGHBOX 0x0080L    /* Box appears around menu item
                      when selected */
HIGHNONE 0x00C0L   /* Menu item does nothing when selected */
CHECKED 0x0100L    /* Checked menu item */
ISDRAWN 0x1000L    /* Drawn menu item */
HIGHITEM 0x2000L   /* Selected menu item */
MENUTOGGLED 0x4000L/* Toggled menu item */
```

6.3.4 Requester and alert functions

AutoRequest — **Creates and processes requester**

Syntax:

```
Response = AutoRequest(Window, BodyText, PositiveText,
    D0         -348       A0       A1           A2
                              NegativeText, PositiveFlags,
                                     A3            D0
                              NegativeFlags, Width, Height);
                                     D1          D2     D3
BOOL Response;
struct Window *Window;
struct IntuiText *BodyText;
struct IntuiText *PositiveText;
struct IntuiText *NegativeText;
ULONG PositiveFlags, NegativeFlags;
SHORT Width, Height;
```

Description: This function creates a requester from the available data and processes the result selected by the user.

Parameters:

Window:	Pointer to window whose input channel should be interrupted.
BodyText:	IntuiText structure for descriptive text.
PositiveText:	IntuiText structure for right gadget text.
NegativeText:	IntuiText structure for left gadget text.
PositiveFlags:	IDCMP flags for display in right gadget.
NegativeFlags:	IDCMP flags for display in left gadget.
Width:	Requester window width.
Height:	Requester window height.

Result:

Response: Returns FALSE when the user clicks on the left gadget and TRUE when the user clicks on the right gadget.

Warning: If insufficient memory exists, this function displays an alert using `DisplayAlert()`.

Comments: The Workbench screen is always in the foreground when a system requester is called.

See Also: `BuildSysRequest(),Request()`

BuildSysRequest — Creates and displays requester

Syntax:

```
ReqWindow = BuildSysRequest(Window, BodyText, PositiveText,
   D0             -360         A0       A1          A2
                           NegativeText, IDCMPFlags, Width, Height);
                                 A3          D0        D2     D3
struct Window *ReqWindow;
struct Window *Window;
struct IntuiText *BodyText;
struct IntuiText *PositiveText;
struct IntuiText *NegativeText;
ULONG IDCMPFlags;
SHORT Width, Height;
```

Description: This function creates a system requester borrowed by `AutoRequest()`. You can also access this system requester and create your own testing loop to access the requester.

Parameters:

Window:	Pointer to window whose input channel should be interrupted.
BodyText:	IntuiText structure for descriptive text.
PositiveText:	IntuiText structure for right gadget text.
NegativeText:	IntuiText structure for left gadget text.
IDCMPFlags:	IDCMP flags for requester window.
Width:	Requester window width.
Height:	Requester window height.

Result: ReqWindow: Pointer to the requester window.

Warning: If insufficient memory exists, this function displays an alert using `DisplayAlert()`.

Comments: Requester gadgets are distinguishable by their IDs: FALSE and TRUE. Both gadgets have the attributes BOOLGADGET, RELVERIFY, REQGADGET, and TOGGLESELECT.

See Also: `AutoRequest(),FreeSysRequest(),DisplayAlert(), ModifyIDCMP()`

ClearDMRequest — Removes double mouse click requester

Syntax:

```
Response = ClearDMRequest(Window);
   D0           -48         A0
BOOL Response;
struct Window *Window;
```

Description: This function clears a double menu request previously set using `SetDMRequest()`.

Parameter: Window: Pointer to window containing defined double menu request.

Result: Response: Returns TRUE if requester is removed and inaccessible, and FALSE in any other cases.

See Also: `SetDMRequest(),Request()`

DisplayAlert — Creates and displays alert

Syntax:

```
Response = DisplayAlert(AlertNumber, String, Height);
    D0          -90         D0          A0      A1
BOOL Response;
ULONG AlertNumber;
UBYTE *String;
SHORT Height;
```

Description: This function displays an alert, moving the screen down by the specified number of lines.

Parameters:

AlertNumber: LONG value number displayed in alert box (highest bit of number displayed only).
String: Pointer to string to be displayed in alert box.
Height: Height of alert box in screen lines.

Result: Response: Returns FALSE after `DEADEND_ALERT`. If a `RECOVER_ALERT` occurs, then the user selection of a mouse button dictates the response: TRUE for the left mouse button, and FALSE for the right mouse button.

Warning: If the system crashes completely it changes the `ALERT_TYPE` into a `DEADEND_ALERT`. The `DEADEND_ALERT` can only be resolved by a system reset.

EndRequest — Removes requester and redisplays window

Syntax:

```
EndRequest(Requester, Window);
   -120       A0         A1
struct Requester *Requester;
struct Window *Window;
```

Description: This function removes the specified requester from the window, and sets the window's status to normal if this was the last requester in a group.

Parameters:

Requester: Pointer to requester structure.
Window: Pointer to window containing the requester.

FreeSysRequest — Releases memory allocated by request

Syntax:

```
FreeSysRequest(Window);
    -372        A0
struct Window *Window;.
```

Description: This function ends a requester added by Intuition and managed by the program. It also releases any memory allocated by Intuition.

Parameter: Window: Pointer to a window used in `BuildSysRequest()` function.

See Also: `BuildSysRequest(),AutoRequest()`

InitRequester — Allocates a requester with general values

Syntax:

```
InitRequester(Requester);
     -138       A0
struct Requester *Requester;
```

Description: This function removes a requester structure and sets the necessary values to zero.

Parameter: Requester: Pointer to requester structure.

See Also: `Request(),EndRequest()`

Structures:

```
struct Requester <intuition/intuition.h>
{
0x00  00  struct Requester *OlderRequester   /* Internal pointer
                                                 to previous
                                                 requester */
0x04  04  SHORT LeftEdge; /* Upper left pixel position of
                             requester in window */
0x06  06  SHORT TopEdge;
0x08  08  SHORT Width;    /* Requester width in pixels */
0x0A  10  SHORT Height;   /* Requester height in pixels */
0x0C  12  SHORT RelLeft;  /* Relative coordinates with a
                             relative position statement to
                             mouse pointer */
0x0E  14  SHORT RelTop;
0x10  16  struct Gadget *ReqGadget; /* Pointer to first
                                       requester gadget */
0x14  20  struct Border *ReqBorder; /* Pointer to first
                                       requester border */
0x18  24  struct IntuiText *ReqText;/* Pointer to first
                                       requester text */
0x1C  28  USHORT Flags;   /* Flag settings for this requester */
0x1E  30  UBYTE BackFill;   /* Requester background color */
0x20  32  struct Layer *ReqLayer;   /* Pointer to layer
                                       structure managed by the
                                       requester */
0x24  36  UBYTE ReqPad1[32];        /* More memory bytes */
0x44  68  struct Bitmap *ImageBMap; /* Pointer to requester
                                       bit-map */
0x48  72  struct Window *RWindow;   /* Pointer to window in
```

```
                                          which the requester
                                          appears */
0x4C  76  UBYTE ReqPad2[36];           /* More memory bytes */
0x70  112
};
```

Requester_Flags:

```
POINTREL 0x0001L     /* Requester appears relative to the mouse
                        pointer */
PREDRAWN 0x0002L
NOISYREQ 0x0004L     /* Not a system requester */
REQOFFWINDOW 0x1000L /* Requester exists outside window */
REQACTIVE 0x2000L    /* Active requester */
SYSREQUEST 0x4000L   /* Handle as system requester */
DEFERREFRESH 0x8000L
```

Request — Activates requester

Syntax:

```
Success = Request(Requester, Window);
  D0        -240      A0        A1
BOOL Success;
struct Requester *Requester;
struct Window *Window;
```

Description: This function opens a predefined requester in the given window.

Parameters:
- Requester: Pointer to requester.
- Window: Pointer to window whose input channel is interrupted.

Result:
- Success: Returns TRUE if the requester can be opened as usual, and FALSE in any other case.

Comments: The POINTREL requester is not currently implemented, but the double menu requesters operate.

See Also: `EndRequest()`

SetDMRequest — Determines DMRequest for window

Syntax:

```
Success = SetDMRequest(Window, DMRequester);
              -258        A0        A1
BOOL Success;
struct Window *Window;
struct Requester *DMRequester;
```

Description: This function defines the double menu requester (a requester which requires a double-click). If a double menu requester is already defined and active, the function does not execute. You must then access `ClearDMRequest()` until TRUE is returned.

Parameters:
- Window: Pointer to window in which DMRequest should be defined.

	DMRequester:	Pointer to request structure representing DMRequest.
Result:	Success:	Returns TRUE if no requester was in use, FALSE if a requester is in use.
See Also:	`ClearDMRequest(),Request(),EndRequest()`	

6.3.5 Screen functions

CloseScreen — **Closes Intuition screen**

Syntax:

```
CloseScreen(Screen);
     -66      A0
struct Screen *Screen;
```

Description: This function closes the specified screen and frees bit-map memory and any parameters used by `OpenScreen()`.

Parameter: Screen: Pointer to screen structure.

Warning: `CloseScreen` does not affect open windows, requesters or menus. Make sure that any windows, requesters and menus are closed before invoking `CloseScreen`.

Comment: If the screen closed was the last screen, Intuition tries to open the Workbench screen.

See Also: `OpenScreen()`

CloseWorkbench — **Closes Workbench**

Syntax:

```
Success = CloseWorkBench();
   D0          -78
BOOL Success;
```

Description: This function closes the Workbench screen.

Result: Success: Returns TRUE if the Workbench can be closed and FALSE if it cannot be closed.

Exceptions: If a window from a program is found on the Workbench screen, this window remains open.

Warning: Be aware of the fact that the control is taken from a program because by closing this screen you have pulled the "floor from under your feet".

See Also: `OpenWorkbench(),CloseScreen()`

DisplayBeep — Blinks screen

Syntax:

```
DisplayBeep(Screen);
    -96      A0
struct Screen *Screen;
```

Description: This function blinks the specified screen.

Parameter: Screen: Pointer to screen structure.

Exceptions: If zero is given as the screen pointer, all available screens blink.

GetScreenData — Gets screen information

Syntax:

```
Success = GetScreenData(Buffer, Size, Type, Screen);
              -426       A0      D0    D1    A1
BOOL Success;
CPTR Buffer;
USHORT Size;
USHORT Type;
struct Screen *Screen;
```

Description: This function gets screen structure data and places the data in a buffer.

Parameters:

Buffer:	Pointer to buffer.
Size:	Data buffer size.
Type:	Type of screen (WBENCHSCREEN, CUSTOMSCREEN).
Screen:	Pointer to screen structure.

Result: Returns TRUE if no error occurs and FALSE if the screen cannot be accessed.

Exceptions: No pointer is needed for the Workbench screen. This is opened if it was previously closed.

MakeScreen — Executes Intuition integrated MakeVPort

Syntax:

```
MakeScreen(Screen);
   -378     A0
struct Screen *Screen;
```

Description: This function executes a `MakeVPort()` of a custom screen through Intuition.

Parameter: Screen: Pointer to screen structure.

See Also: `RethinkDisplay(),RemakeDisplay()`

MoveScreen **Moves the screen by given delta**

Syntax:

```
MoveScreen(Screen, DeltaX, DeltaY);
   -162      A0      D0      D1
struct Screen *Screen;
SHORT DeltaX, DeltaY;
```

Description: This function moves the screen by the specified delta values. The distance is relative to the current location rather than by absolute coordinates.

Parameters:

Screen: Pointer to screen.
DeltaX: Horizontal movement value.
DeltaY: Vertical movement value.

Exceptions: In the current version of the operating system the screen can only be moved in the vertical direction. For compatibility reasons the `DeltaX` value should always be set to zero.

OpenScreen **Opens Intuition screen**

Syntax:

```
Screen = OpenScreen(NewScreen);
   D0       -198        A0
struct Screen *Screen;
struct NewScreen *NewScreen;
```

Description: This function opens a new screen from the `NewScreen` structure. It supplies the pointer to an added screen structure. Other operations can be done with this pointer.

Parameter: NewScreen: Pointer to `NewScreen` structure.

Result: Screen: Pointer to the newly added screen structure.

See Also: `CloseScreen(),MakeScreen()`

```
struct NewScreen <intuition/intuition.h>
{
0x00  00  SHORT LeftEdge;   /* left border of screen relative to
                               the View (not used) */
0x02  02  SHORT TopEdge;    /* Top of screen relative to the
                               View */
0x04  04  SHORT Width;      /* Screen width and height */
0x06  06  SHORT Height;
0x08  08  SHORT Depth;      /* Screen depth (changes with color
                               number) */
0x0A  10  UBYTE DetailPen;  /* Detail pen color number */
0x0B  11  UBYTE BlockPen;   /* Block pen color number */
0x0C  12  USHORT ViewModes; /* View mode flags */
0x0E  14  USHORT Type; /* Screen type (Workbench/Custom) */
0x10  16  struct TextAttr *Font;  /* Pointer to screen font */
0x14  20  UBYTE *DefaultTitle;    /* Pointer to default screen
                                     title */
0x18  24  struct Gadget *Gadgets; /* Pointer to screen gadgets
```

```
                                   (not used) */
0x1C  28  struct Bitmap *CustomBitmap; /* Pointer to added
                                      screen bit-map */
0x20  32
};

struct Screen <intuition/intuition.h>
{
0x000  00  struct Screen *NextScreen;  /* Pointer next screen's
                                      list */
0x004  04  struct Window *FirstWindow; /* Pointer to screen's
                                      first window */
0x008  08  SHORT LeftEdge;  /* Left border relative to the View
                            (not used) */
0x00A  10  SHORT TopEdge;  /* Top border relative to the View */
0x00C  12  SHORT Width;  /* Screen width and height in pixels */
0x00E  14  SHORT Height;
0x010  16  SHORT MouseY; /* Mouse coordinates on screen */
0x012  18  SHORT MouseX;
0x014  20  USHORT Flags; /* Screen flags */
0x016  22  UBYTE *Title; /* Pointer to screen title text as a
                          string */
0x01A  26  UBYTE *DefaultTitle; /* Pointer to default title
                                text */
0x01E  30  BYTE BarHeight;     /* Height of the title bar in
                               pixels */
0x01F  31  BYTE BarVBorder;     /* Vertical width of title bar
                                border */
0x020  32  BYTE BarHBorder;     /* Horizontal border width */
0x021  33  BYTE MenuVBorder;    /* Menu border width of
                                (vertical/horizontal) */
0x022  34  BYTE MenuHBorder;
0x023  35  BYTE WBorTop;    /* Window border width (top) */
0x024  36  BYTE WBorLeft;   /* (left) */
0x025  37  BYTE WBorRight;  /* (right) */
0x026  38  BYTE WBorBottom; /* (bottom) */
0x028  40  struct TextAttr *Font;   /* Pointer to screen font */
0x02C  44  struct ViewPort ViewPort; /* Pointer to ViewPort */
0x054  84  struct RastPort RastPort; /* Connecting the RastPort
                                    structure */
0x0B8 184  struct Bitmap Bitmap;       /* Connecting the bit-map
                                      structure */
0x0E0 224  struct Layer_Info LayerInfo; /* Connecting the
                                      layerinfo structure */
0x13C 316  struct Gadget *FirstGadget;  /* Pointer to first
                                       screen gadget (not
                                       supported) */
0x140 320  UBYTE DetailPen    /* Detail pen color number */
0x141 321  UBYTE BlockPen;    /* Block pen color number */
0x142 322  USHORT SaveColor0; /* Buffer memory for screen color
                              0 when blinking */
0x144 324  struct Layer *BarLayer;/* Pointer to title bar layer
when presenting the menu item */
0x148 328  UBYTE *ExtData;        /* Pointer to more data
                                  (Expansion) */
0x14C 332  UBYTE *UserData;       /* Pointer to user data of
                                  this screen */
0x150 336
};
```

Screen_Flags:

```
SCREENTYPE 0x000FL
WBENCHSCREEN 0x0001L /* Workbench screen ID*/
CUSTOMSCREEN 0x000FL /* Every other screen is a CustomScreen */
SHOWTITLE 0x0010L    /* Title bar drawn */
BEEPING 0x0020L      /* Screen blinks */
CUSTOMBITMAP 0x0040L /* User bit-map */
SCREENBEHIND 0x0080L /* Screen opens behind all others */
SCREENQUIET 0x0100L  /* Screen has no gadgets/menu strip */
STDSCREENHEIGHT -1L  /* Default screen height (200 pixels) */
```

OpenWorkBench — Opens Workbench

Syntax:

```
WBScreen = OpenWorkBench();
   D0          -210
struct Screen *WBScreen;
```

Description: This function opens the Workbench screen and displays all Workbench icons and windows.

Result: WBScreen: Pointer to Workbench screen structure.

Exception: Returns FALSE if not enough memory exists.

Warning: Avoid using the pointer, because the Workbench screen can be affected or even closed by external programs.

See Also: `CloseWorkBench(),OpenScreen(),CloseScreen()`

ScreenToBack — Moves screen to background

Syntax:

```
ScreenToBack(Screen);
      -246     A0
struct Screen *Screen;
```

Description: This function places the screen of the given screen structure in the background.

Parameter: Screen: Pointer to screen structure.

See Also: `ScreenToFront(),WBenchToBack(),WBenchToFront()`

ScreenToFront — Moves screen to foreground

Syntax:

```
ScreenToFront(Screen);
    -252       A0
struct Screen *Screen;
```

Description: This function places the screen of the given screen structure in the foreground.

Parameter: Screen: Pointer to screen structure.

See Also: `ScreenToBack(),WBenchToBack(),WBenchToFront()`

ShowTitle — Sets screen title bar display

Syntax:

```
ShowTitle(Screen, ShowIt)
   -282      A0      D0
struct Screen *Screen;
BOOL ShowIt;
```

Description: This function sets the display mode of the screen title bar. The screen title bar covered by a BACKDROP window can be placed in front of or behind the window.

Parameters:

Screen:	Pointer to screen structure.
ShowIt:	Returns TRUE for overlay and FALSE for not drawing.

WBenchToBack — Moves Workbench to background

Syntax:

```
WBenchToBack();
-336
```

Description: This function places the Workbench behind all other screens.

Comments: The function can be called by pressing the <right Amiga><M> key combination.

See Also: `WBenchToFront(),ScreenToBack(),ScreenToFront()`

WBenchToFront — Moves Workbench to foreground

Syntax:

```
WBenchToFront();
-342
```

Description: This function places the Workbench in front of all other screens.

Comments: Invoking a requester or pressing <right Amiga><N> executes this function.

See Also: `WBenchToBack(),ScreenToBack(),ScreenToFront()`

6.3.6 Graphic functions

ClearPointer **Clears mouse pointer**

Syntax:

```
ClearPointer(Window);
    -60         A0
struct Window *Window;
```

Description: This function clears the custom mouse pointer from the window. After the call the mouse cursor appears as set under Preferences.

Parameter:

Window: Pointer to the window in which the mouse pointer should be changed to the default graphic.

See Also: `SetPointer()`

DrawBorder **Draws border structure in RastPort**

Syntax:

```
DrawBorder(RastPort, Border, LeftOffset, RightOffset);
   -108       A0       A1       D0            D1
struct RastPort *RastPort;
struct Border *Border;
SHORT LeftOffset, RightOffset;
```

Description: This function draws the given lines in the RastPort at the position specified by the offsets. If the `NextBorder` array of the structure contains more data, these lines are also drawn.

Parameters:

RastPort: Pointer to RastPort to receive the new border.
Border: Border structure defining lines.
LeftOffset: Offset value added to each X coordinate.
TopOffset: Offset value added to each Y coordinate.

See Also: `DrawImage(),PrintIText()`

Structure:

```
struct Border <intuition/intuition.h>
{
0x00  00  SHORT LeftEdge  /* Pixel position of border
                             relative to the RastPort */
0x02  02  SHORT TopEdge;
0x04  04  UBYTE FrontPen  /* Line drawing color */
0x05  05  UBYTE BackPen;  /* Not used */
0x06  06  UBYTE DrawMode; /* Draw mode=JAM1 */
0x07  07  BYTE Count;     /* Number of coordinate pairs */
0x08  08  SHORT *XY;      /* Coordinate table pointer to */
0x0C  12  struct Border *NextBorder; /* Link to other
                                        border structures */
0x10  16
};
```

DrawImage — Draws image in RastPort

Syntax:
```
DrawImage(RastPort, Image, LeftOffset, TopOffset);
   -114      A0        A1        D0          D1
struct RastPort *RastPort;
struct Image *Image;
SHORT LeftOffset, TopOffset;
```

Description: This function draws the given image in the RastPort at the position specified by the offsets. If the `NextImage` array contains more data, these images are also drawn.

Parameters:

RastPort:	Pointer to RastPort to receive the new image.
Image:	Image structure defining image.
LeftOffset:	Offset value added to each X coordinate.
TopOffset:	Offset value added to each Y coordinate.

See Also: `DrawBorder(),PrintIText()`

Structure:
```
struct Image <intuition/intuition.h>
{
0x00  00  SHORT LeftEdge;     /* Pixel position of the graphic
                                 relative to the RastPort */
0x02  02  SHORT TopEdge;
0x04  04  SHORT Width;   /* Graphic width and height in pixels */
0x06  06  SHORT Height;
0x08  08  SHORT Depth;        /* depth of graphic bit-planes */
0x0A  10  USHORT *ImageData;  /* Pointer to image data */
0x0E  14  UBYTE PlanePick     /* Marking bit-planes
                                 for picking */
0x0F  15  UBYTE PlaneOnOff;   /* Marking bit-planes for
                                 turning off */
0x10  16  struct Image *NextImage; /* Link to more image
                                      structures */
0x14  20
};
```

IntuiTextLength — Returns pixel width of an IntuiText

Syntax:
```
Width = IntuiTextLength(IText);
              -330        D0
USHORT Width;
struct IntuiText *IText;
```

Description: This function calculates the pixel width of an `IntuiText`, independently of the given character set.

Parameter: IText: Pointer to `IntuiText` structure.

Result: Width: Text width in pixels.

See Also: `PrintIText()`

PrintIText — Writes text in RastPort

Syntax:

```
PrintIText(RastPort, IText, LeftOffset, TopOffset);
  -216      A0        A1      D0          D1
struct RastPort *RastPort;
struct IntuiText *IText;
SHORT LeftOffset, TopOffset;
```

Description: This function writes the text of the `IntuiText` structure in the given RastPort at the position specified through the offsets. If the `NextText` array contains more data, this data is also written.

Parameters:

RastPort: Pointer to RastPort to receive the new image.
IText: `IntuiText` structure containing the text.
LeftOffset: Offset value added to each X coordinate.
TopOffset: Offset value added to each Y coordinate.

See Also: `IntuiTextLength(),DrawBorder(),DrawImage()`

Structure:

```
struct IntuiText <intuition/intuition.h>
{
0x00  00  UBYTE FrontPen; /* Number of foreground drawing
                             color */
0x01  01  UBYTE BackPen;  /* Number of background drawing
                             color */
0x02  02  UBYTE DrawMode; /* Draw mode: JAM1, JAM2,
                             COMPLEMENT, INVERSEVID */
0x04  04  SHORT LeftEdge; /* Pixel position of text relative to
                             RastPort */
0x06  06  SHORT TopEdge;
0x08  08  struct TextAttr *ITextFont; /* Pointer to character
                                         set: Null = default */
0x0C  12  UBYTE *IText;               /* Pointer to string */
0x10  16  struct IntuiText *NextText; /* Link to additional
                                         IntuiText structures */
0x14  20
};
```

SetPointer — Defines custom mouse pointer for window

Syntax:

```
SetPointer(Window, Pointer, Height, Width, XOffset, YOffset);
   -270     A0       A1       D0      D1      D2       D3
struct Window *Window;
USHORT *Pointer;
SHORT Height, Width;
SHORT XOffset, YOffset;
```

Description: This function defines a custom mouse pointer for the given window. This is always represented when the window is active. The offsets specify the pointer's position and hot spot.

Parameters:

Window:	Pointer to window structure.
Pointer:	Pointer to the mouse pointer sprite data.
Height:	Sprite height data.
Width:	Sprite width data (<=16).
XOffset:	X offset of sprite's hot spot.
YOffset:	Y offset of sprite's hot spot.

See Also: `ClearPointer()`

6.3.7 Memory functions

AllocRemember — **Allocates memory**

Syntax:

```
MemBlock = AllocRemember(RememberKey, Size, Flags);
    D0          -396          A0         D0     D1
CPTR MemBlock;
struct Remember *RememberKey;
ULONG Size;
ULONG Flags;
```

Description: This function allocates memory using the `AllocMem()` function of the Exec library. In addition, it manages a list which allows easy release of all allocated memory.

Parameters:

RememberKey:
Pointer to `Remember` structure. The pointer must be set to zero on the initial call.

Size: Memory block size.

Flags: Attributes of the desired memory block.

Result:

MemBlock: Returns the pointer to the desired memory block, or zero if the function cannot be executed.

See Also: `FreeRemember(),AllocMem(),FreeMem()`

Structure:

```
struct Remember <intuition/intuition.h>
{
0x00  00  struct Remember *NextRemember;/* Pointer to next
                                        remember structure */
0x04  04  ULONG RememberSize; /* Memory range size */
0x08  08  UBYTE *Memory;      /* Pointer to memory range */
0x0C  12
};
```

FreeRemember **frees the noted memory in the list**

Syntax:

```
FreeRemember(RememberKey, ReallyForget);
    -408         A0             D0
struct Remember *RememberKey;
BOOL ReallyForget;
```

Description: This function frees all of the memory regions that are entered in the `Link_list` from `RememberKey`. You can also clear the `RememberKey` structure through this function.

Parameters:

RememberKey: Pointer to first remember structure of a list.

ReallyForget: Tests for release of just the structure or just the memory range. TRUE releases both the memory and the structure.

See Also: `AllocRemember(),AllocMem(),FreeMem()`

6.3 8 Refresh functions

BeginRefresh **Sets a window for optimum refresh**

Syntax:

```
BeginRefresh(Window);
     -354      A0
struct Window *Window;
```

Description: This routine prepares the given window for redrawing only in areas that require refresh.

Parameter: Window: Pointer to the window.

See Also: `EndRefresh()`

EndRefresh **Disables optimum refresh**

Syntax:

```
EndRefresh(Window, Complete);
   -366      A0      D0
struct Window *Window;
BOOL Complete;
```

Description: This window disables the status enabled by `BeginRefresh`.

Parameters:

Window: Pointer to the window containing the `BeginRefresh` status.

Complete: Truth value that describes whether the refresh can be released.

See Also: `BeginRefresh()`

RemakeDisplay	**Redraws entire Intuition display**

Syntax:
```
RemakeDisplay()
-384
```

Description: This function calls `MakeScreen()` and `Rethinkdisplay()` for all screens, redrawing all display elements controlled through Intuition.

Warning: This function can take some time to execute—use this function sparingly. If `RethinkDisplay()`, `Forbid()` and `Permit()` are called several times, the multitasking system may slow down radically.

See Also: `MakeScreen()`, `RethinkDisplay()`, `MakeVPort()`

RethinkDisplay	**Redraws Intuition display**

Syntax:
```
RethinkDisplay()
-390
```

Description: This function works through all of the ViewPorts and selects the error, corrects it and re-initializes the Copper lists.

Warning: This function can take some time to execute—use this function sparingly. If `RethinkDisplay()`, `Forbid()` and `Permit()` are called several times, the multitasking system may slow down radically.

See Also: `RemakeDisplay()`, `MakeVPorty()`, `MakeScreen()`

6.3.9 Other functions

CurrentTime	**Returns current time value**

Syntax:
```
CurrentTime(Second, Micros);
   -84          A0     A1
ULONG *Seconds, *Micros;
```

Description: This function copies the current values of the system time into the specified memory locations. This time is correct about 60 times per second.

Parameters:

Seconds: Pointer to a LONG variable into which the seconds are entered.

Micros: Pointer to a LONG variable into which the microseconds are entered.

DoubleClick — Tests for two clicks within a time span

Syntax:

```
Is = DoubleClick(StartSecs, StartMicros, CurrentSecs,
D0       -102        D0           D1            D2
                                          CurrentMicros);
                                                D3
BOOL IsDouble;
LONG StartSecs, StartMicros;
LONG CurrentSecs, CurrentMicros;
```

Description: This function tests whether two clicks occur within the time span specified under Preferences.

Parameters:

StartSecs: Time of the first click in seconds.
StartMicros: Time of the first click in microseconds.
CurrentSecs: Time of the second click in seconds.
CurrentMicros: Time of the second click in microseconds.

Result:

IsDouble: Returns TRUE if the time span corresponds to a double-click.

See Also: `CurrentTime()`

GetDefPrefs — Copies Preferences settings to buffer

Syntax:

```
Prefs = GetDefPrefs(PrefBuffer, Size);
  D0        -126        A0        D0
struct Preferences *Prefs;
struct Preferences *PrefBuffer;
SHORT Size;
```

Description: This function copies the default values from Preferences into the specified buffer. These are the values set when the system is started. Other values are searched for on the disk when it is first booted.

Parameters:

PrefBuffer: Pointer to Preferences data buffer.
Size: Buffer size.

Result:

Prefs: Pointer to the buffer in which the data is placed (usually the same as the Preferences buffer).

See Also: `GetPrefs()`

GetPrefs — Returns current Preferences settings

Syntax:

```
Prefs = GetPrefs(PrefBuffer, Size);
  D0      -132      A0        D0
struct Preferences *Prefs;
struct Preferences *PrefBuffer;
SHORT Size;
```

Description: This function copies the settings from Preferences into the given buffer. These are the values set by the user through the Preferences program.

Parameters:

PrefBuffer:	Pointer to Preferences data buffer.
Size:	Buffer size.

Result:

Prefs:	Pointer to the buffer in which the data is placed (usually the same as the Preferences buffer).

See Also: `GetPrefs()`

Structure:

```
struct Preferences <intuition/intuition.h>
{
0x00  00  BYTE FontHeight;   /* character set: 60 or 80
                                characters */
0x01  01  UBYTE PrinterPort; /* PrinterPort: serial or
                                parallel */
0x02  02  USHORT BaudRate; /* BaudRate: between 110 and 19200 */
0x04  04  struct timeval KeyRptSpeed; /* Keyboard repeat
                                         speed */
0x0C  12  struct timeval KeyRptDelay; /* Delay time until a
                                         key repeat */
0x14  20  struct timeval DoubleClick; /* Double-click
                                         time interval */
0x1C  28  USHORT PointerMatrix[36L];  /* Mouse pointer graphic
                                         data */
0x64 100  BYTE XOffset;         /* Hot spot offset */
0x65 101  BYTE YOffset;
0x66 102  USHORT color17;       /* Mouse pointer colors */
0x68 104  USHORT color18;
0x6A 106  USHORT color19;
0x6C 108  USHORT PointerTicks;  /* Mouse movement conversion */
0x6E 110  USHORT color0;        /* Workbench colors */
0x70 112  USHORT color1;
0x72 114  USHORT color2;
0x74 116  USHORT color3;
0x76 118  BYTE ViewXOffset;   /* Relative position of the
                                 Workbench screen to the View */
0x77 119  BYTE ViewYOffset;
0x78 120  WORD ViewInitX;       /* Initialization values for the
                                   View */
0x7A 122  WORD ViewInitY;
0x7C 124  BOOL EnableCLI;       /* Enable/disable CLI */
0x7E 126  USHORT PrinterType;   /* Printer type */
0x80 128  UBYTE PrinterFilename[30L]; /* Name of the printer
                                         with CUSTOM */
0x9E 158  USHORT PrintPitch;    /* Kind of type: Pica, Elite,
                                   Fine */
0xA0 160  USHORT PrintQuality;  /* Print quality: Draft, NLQ */
0xA2 162  USHORT PrintSpacing;  /* Print spacing: 6 LPI or 8
                                   LPI */
0xA4 164  UWORD PrintLeftMargin; /* Left and right print
                                   margin */
0xA6 166  UWORD PrintRightMargin;
0xA8 168  USHORT PrintImage;    /* Positive or negative graphic
                                   imaging */
0xAA 170  USHORT PrintAspect;   /* Print aspect: horizontal,
                                   vertical */
```

```
0xAC 172  USHORT PrintShade;      /* Print type: black/white,
                                     gray shade, color */
0xAE 174  WORD PrintThreshold;    /* Gray scaling */
0xB0 176  USHORT PaperSize;       /* Paper size: Flags */
0xB2 178  UWORD PaperLength;      /* Paper length in lines */
0xB4 180  USHORT PaperType;       /* Paper type: continuous feed,
                                     single sheet */
0xB6 182  UBYTE SerRWBits;        /* Serial settings:
                                     Read/Write Bits */
0xB7 183  UBYTE SerStopBuf;       /* number of stop bits, buffer
                                     size */
0xB8 184  UBYTE SerParShk;        /* Parity and Shake */
0xB9 185  UBYTE LaceWB;           /* Interlace Mode:
                                     on, off */
0xBA 186  UBYTE WorkName[30L];    /* Buffer storage of
                                     printer name */
0xD8 216  BYTE RowSizeChange;
0xD9 217  BYTE ColumnSizeChange;  /* Final Version 1.2 */
0xDA 218  UWORD PrintFlags;       /* New graphic settings  */
0xDC 220  UWORD PrintMaxWidth;
0xDE 222  UWORD PrintMaxHeight;
0xE0 224  UBYTE PrintDensity;
0xE1 225  UBYTE PrintXOffset;
0xE2 226  UWORD wb_Width;         /* Workbench width, height,
                                     depth */
0xE4 228  UWORD wb_Height;
0xE6 230  UBYTE wb_Depth;         /* Version 1.3 */
0xE7 231  UBYTE ext_size;         /* Length of an integrated
                                     expansion */
0xE8 232
};
```

Preferences_FontHeight:

```
TOPAZ_EIGHTY 8L
TOPAZ_SIXTY 9L
```

Preferences_LaceWB:

```
LACEWB 0x01L
```

Preferences_PrinterPort:

```
PARALLEL_PRINTER 0x00L
SERIAL_PRINTER 0x01L
```

Preferences_BaudRate:

```
BAUD_110 0x00L
BAUD_300 0x01L
BAUD_1200 0x02L
BAUD_2400 0x03L
BAUD_4800 0x04L
BAUD_9600 0x05L
BAUD_19200 0x06L
BAUD_MIDI 0x07L
```

Preferences_PaperType:

```
FANFOLD 0x00L
SINGLE 0x80L
```

Preferences_PrintPitch:

```
PICA 0x000L
ELITE 0x400L
FINE 0x800L
```

Preferences_PrintQuality:

```
DRAFT 0x000L
LETTER 0x100L
```

Preferences_PrintSpacing:

```
SIX_LPI 0x000L
EIGHT_LPI 0x200L
```

Preferences_PrintImage:

```
IMAGE_POSITIVE 0x00L
IMAGE_NEGATIVE 0x01L
```

Preferences_PrintAspect:

```
ASPECT_HORIZ 0x00L
ASPECT_VERT 0x01L
```

Preferences_PrintShade:

```
SHADE_BW 0x00L
SHADE_GREYSCALE 0x01L
SHADE_COLOR 0x02L
```

Preferences_PaperSize:

```
US_LETTER 0x00L
US_LEGAL 0x10L
N_TRACTOR 0x20L
W_TRACTOR 0x30L
CUSTOM 0x40L
```

Preferences_PrinterType:

```
CUSTOM_NAME 0x00L
ALPHA_P_101 0x01L
BROTHER_15XL 0x02L
CBM_MPS1000 0x03L
DIAB_630 0x04L
DIAB_ADV_D25 0x05L
DIAB_C_150 0x06L
EPSON 0x07L
EPSON_JX_80 0x08L
OKIMATE_20 0x09L
```

```
QUME_LP_20 0x0AL
HP_LASERJET 0x0BL
HP_LASERJET_PLUS 0x0CL
```

Preferences_SerialBuffer:

```
SBUF_512 0x00L
SBUF_1024 0x01L
SBUF_2048 0x02L
SBUF_4096 0x03L
SBUF_8000 0x04L
SBUF_16000 0x05L
```

Preferences_SerRWBits:

```
SREAD_BITS 0xF0L
SWRITE_BITS 0x0FL
```

Preferences_SerStopBuf:

```
SSTOP_BITS 0xF0L
SBUFSIZE_BITS 0x0FL

Preferences_SerParShk
SPARITY_BITS 0xF0L
SPARITY_NONE 0L
SPARITY_EVEN 1L
SPARITY_ODD 2L
SHSHAKE_XON 0L
SHSHAKE_RTS 1L
SHSHAKE_NONE 2L
```

LockIBase — Returns an Intuition lock

Syntax:

```
Lock = LockIBase(LockNumber);
 D0       -414        D0
ULONG Lock;
ULONG LockNumber;
```

Description: This function locks the entire Intuition system. The function is called from Intuition by any elementary change made to structures.

Parameter: LockNumber: Number of the lock. A value of zero indicates a lock that frees all of the elements for searching.

Result: Lock: Lock number used by `UnlockIbase()`.

Warning: The entire Intuition system waits until `UnlockIBase()` is called.

Comments: This function cannot be called if another lock is currently executing (`LayerInfo`).

See Also: `UnlockIBAse()`, `LockLayerInfo()`, `ObtainSemaphore()`

ReportMouse — Toggles mouse movement reports

Syntax:

```
ReportMouse(Boolean, Window);
    -234      A0       D0
BOOL Boolean;
struct Window *Window;
```

Description: This function toggles the REPORTMOUSE flag contained in the window's IDCMP. If the mouse clicks on a gadget, this sets the FOLLOWMOUSE flag.

Parameters:

Window: Pointer to the window.
Boolean: Truth value which differentiates whether the flag should be set (TRUE) or cleared (FALSE).

Exceptions: The AZTEC C compiler calls the function using the syntax ReportMouse(Window, (ULONG)Boolean);.

SetPrefs — Sets Preferences

Syntax:

```
Prefs = SetPrefs(PrefBuffer, Size, Inform);
  D0       -324     A0         D0     D1
struct Preferences *Prefs;
struct Preferences *PrefBuffer;
int Size;
BOOL Inform;
```

Description: This function copies the given number of bytes of the Preference structure into the system preferences table. Inform can add other items if NEWPREFS informs the program of this.

Parameters:

PrefBuffer: Pointer to the buffer that contains the new data.
Size: Number of bytes that should be copied.
Inform: Truth value which differentiates if NEWPREFS is set (TRUE) or cleared (FALSE).

Result:

Prefs: Pointer to the data buffer.

See Also: GetDefPrefs(),GetPrefs()

UnlockIBase — Frees Intuition lock

Syntax:

```
UnlockIBase(Lock);
    -420     A0
ULONG Lock;
```

Description: This function frees the lock actuated using LockIBase().

Parameter:

Lock: Value returned by LockIBase().

Warning: If you try to free a lock that that doesn't exist, the system crashes.

See Also: `LockIBase()`

ViewAddress — Supplies View structure address

Syntax:

```
Address = ViewAddress();
                -294
struct View *Address;
```

Description: This function returns the pointer to the View, needed for every graphic operation.

Result: Address: Intuition View address.

ViewPortAddress — Supplies ViewPort structure address

Syntax:

```
Address = ViewPortAddress(Window);
                -300          A0
struct View *Address;
struct Window *Window;.
```

Description: This function returns the pointer to the ViewPort, needed for every graphic operation performed within a window.

Parameter: Window: Pointer to a window structure.

Result: Address: ViewPort address of this window.

More Intuition Structures:

```
struct IntuiMessage <intuition/intuition.h>
{
0x00  00  struct Message ExecMessage; /* ExecMessage structure
                                          integration */
0x14  20  ULONG Class;      /* Report classification */
0x18  24  USHORT Code;      /* used for more values */
0x1A  26  USHORT Qualifier; /* Identifies the keyboard plane */
0x1C  28  APTR IAddress;    /* Pointer to an Intuition object
                               released by the report */
0x20  32  SHORT MouseX;     /* Current mouse position in
                               pixels, relative to the
                               window */
0x22  34  SHORT MouseY;
0x24  36  ULONG Seconds;    /* Clock time of report */
0x28  40  ULONG Micros;
0x2C  44  struct Window *IDCMPWindow;     /* Pointer to the
                                             IDCMPWindow that
                                             sends the report */
0x30  48  struct IntuiMessage *SpecialLink; /* Pointer to the
                                               next IntuiMessage
                                               Structure */
0x34  52
```

IntuiMessage_IDCMPFlags:

```
SIZEVERIFY 0x00000001L    /* Changed window size */
NEWSIZE 0x00000002L
REFRESHWINDOW 0x00000004L /* Redraw the window */
MOUSEBUTTONS 0x00000008L  /* Mouse button pressed */
MOUSEMOVE 0x00000010L     /* Mouse was moved */
GADGETDOWN 0x00000020L    /* Gadget pressed down */
GADGETUP 0x00000040L      /* Gadget released */
REQSET 0x00000080L        /* Requester added to window */
MENUPICK 0x00000100L      /* Menu item selected */
CLOSEWINDOW 0x00000200L   /* Window closed */
RAWKEY 0x00000400L        /* RAWKEY report sent */
REQVERIFY 0x00000800L     /* Requester should be checked
                             more closely */
REQCLEAR 0x00001000L      /* Requester erased again */
MENUVERIFY 0x00002000L    /* Verification before menu
                             appears */
NEWPREFS 0x00004000L      /* New Preferences added */
DISKINSERTED 0x00008000L  /* Disk inserted in a drive */
DISKREMOVED 0x00010000L   /* Disk removed from a drive */
WBENCHMESSAGE 0x00020000L /* Handle as report from
                             the WorkBench */
ACTIVEWINDOW 0x00040000L  /* Window activated */
INACTIVEWINDOW 0x00080000L/* Window deactivated */
DELTAMOVE 0x00100000L     /* Mouse coordinates relative
                             to the last position */
VANILLAKEY 0x00200000L    /* Handle data as unprocessed
                             keyboard reports */
INTUITICKS 0x00400000L    /* Clock time transferred */
LONELYMESSAGE 0x80000000L /* No Message Type */
```

Menu_Flags:

```
MENUHOT 0x0001L           /* Menu Status */
MENUCANCEL 0x0002L
MENUWAITING 0x0003L
OKOK MENUHOT
OKABORT 0x0004L
OKCANCEL MENUCANCEL
```

Workbench_Flags:

```
WBENCHOPEN 0x0001L             /* Workbench was opened */
WBENCHCLOSE 0x0002L            /* Workbench was closed */
```

Intuition_Macros:

Menu_Macros:

```
MENUNUM(n)  (n & 0x1F)            /* Menu number */
ITEMNUM(n)  ((n >> 5) & 0x003F)   /* Number of menu items */
SUBNUM(n)  ((n >> 11) & 0x001F)   /* Number of submenu items */
SHIFTMENU(n)  (n & 0x1F)          /* Shifted menu number */
SHIFTITEM(n)  ((n & 0x3F) << 5)   /* Number of shifted menu
                                     items */
SHIFTSUB(n)  ((n & 0x1F) << 11)   /* Number of shifted submenu
                                     items */
```

Serial_Macros:

```
SRBNUM(n)  (0x08        (n >> 4))       /* Number of read bits */
SWBNUM(n)  (0x08        (n & 0x0F))     /* Number of write bits */
SSBNUM(n)  (0x01 + (n >> 4)).
SPARNUM(n)  (n >> 4)
SHAKNUM(n)  (n & 0x0F)
```

Preferences_Definition:

```
NOMENU 0x001FL
NOITEM 0x003FL
NOSUB 0x001FL
MENUNull 0xFFFFL
FOREVER for(;;)                  /* Infinite loop */
SIGN(x) ( ((x) > 0)    ((x) < 0) )
NOT !
CHECKWIDTH 19L
COMMWIDTH 27L
LOWCHECKWIDTH 13L
LOWCOMMWIDTH 16L
ALERT_TYPE 0x80000000L           /* Alert mask */
RECOVERY_ALERT 0x00000000L       /* Recoverable error */
DEADEND_ALERT 0x80000000L        /* Non-recoverable alert */
AUTOFRONTPEN 0L                  /* Standard drawing colors */
AUTOBACKPEN 1L
AUTODRAWMODE JAM2                /* Standard drawing mode */
AUTOLEFTEDGE 6L                  /* Standard coordinates */
AUTOTOPEDGE 3L
AUTOITEXTFONT Null               /* Standard left */
AUTONEXTTEXT Null
SELECTUP (IECODE_LBUTTON | IECODE_UP_PREFIX)
SELECTDOWN (IECODE_LBUTTON)
MENUUP (IECODE_RBUTTON | IECODE_UP_PREFIX)
MENUDOWN (IECODE_RBUTTON)
ALTLEFT (IEQUALIFIER_LALT)
ALTRIGHT (IEQUALIFIER_RALT)
AmigaLEFT (IEQUALIFIER_LCOMMAND)
AmigaRIGHT (IEQUALIFIER_RCOMMAND)
AmigaKEYS (AmigaLEFT | AmigaRIGHT)
CURSORUP 0x4CL
CURSORLEFT 0x4FL
CURSORRIGHT 0x4EL
CURSORDOWN 0x4DL
KEYCODE_Q 0x10L
KEYCODE_X 0x32L
KEYCODE_N 0x36L
KEYCODE_M 0x37L
KEYCODE_V 0x34L
KEYCODE_B 0x35L
```

6.4 The layers library

Layers are rectangular graphic objects which can be overlapped. Each layer has its own `RastPort` through which it can activate graphic operations in the layer. Before the layers can be used the layers library must be opened:

```
Long *LayersBase;
..
layersBase = OpenLibrary ("layers.library",0);
```

In addition, you should open the graphics library because it contains important functions used for creating clipping rectangles. Clipping rectangles are layer sections. Graphic operations draw only the insides of these clipping rectangles. Layers can be moved on the screen and vary in size.

Intuition windows are represented by layers. Almost all of the attributes used by windows are also used by the layers library.

6.4.1 Layer creation

6.4.2 Layer processing

6.4.3 Releasing layers

6.4.1 Layer creation

CreateBehindLayer — **Creates background layer**

Syntax:

```
Layer = CreateBehindLayer (LayerInfo, Bit-map, x1, y1,x2, y2,
  D0            -42               A0       A1    D0  D1  D2 D3,
                                          Flags [,Superbitmap])
                                           D4        [ A2 ]

struct Layer *Layer;
struct Layer_Info *LayerInfo;
struct Bit-map *Bit-map;
LONG x1,y1,
x2,y2;
LONG Flags;
[struct Bit-map *Superbitmap;]
```

Description: This function creates a layer which is placed behind other available layers.

Parameters:

LayerInfo: Address of a completely initialized LayerInfo structure

Bit-map: Address of the bit-map in which the layer should be displayed.

x1, y1, x2, y2:
Upper left and lower right corners of the layer.

Flags: Describe layer type.

LayerSimple: Flag to test whether disturbed layer section should be refreshed.

Layersmart: A layer section overlapped by other layers is stored in a memory range allocated by the system for easy refresh. If insufficient memory exists, a Guru Meditation occurs.

Layersuper: Superbitmap support. Superbitmaps are bit-maps that are too large to appear on the screen. This superbitmap must be allocated by you but not displayed. The layer draws a normal bit-map, and the superbitmap is transferred to the layer containing the normal bit-map.

If you draw outside of the layer's border, these lines are transferred directly into the superbitmap. If `SizeLayer()` or `ScrollLayer()` are used, the normal bit-map is copied into the correct position of the superbitmap, and then the new section of the superbitmap is copied back and displayed. When enlarging the layer (`SizeLayer`) the section of the layering coming back is directly displayed.

Backdrop: Operates in conjunction with all other flags (`LAYERSIMPLE`, `LAYERSMART` and `LAYERSUPER` cancel out other flags). A BACKDROP layer stands behind all other layers using `CreateBehindLayer()`, or in front of all of the other BACKDROPS and behind the rest of the layers using `CreateUpFrontLayers()`.

SuperBitmap: Address of the superbitmap that must be presented in conjunction with the `LAYERSUPER` flag.

Layer: Address of a completely initialized layer structure.

Comments: Certain flags must be set in conjunction with certain layer controls. For example, if you want to refresh a `LAYERSIMPLE` layer, the `LAYERREFRESH` flag must be set in `Layer->Flags`. This and the other flags are defined in the include file `graphics/layer.h`. When using the `LAYERSUPER` flag you should also set the `LAYERSMART` flag.

See Also: `CreateUpFrontLayer()`

CreateUpFrontLayer — Creates foreground layer

Syntax:

```
Layer = CreateUpfrontLayer (LayerInfo, Bit-map, x1, y1,
  D0          -36               A0         A1    D0  D1
x2, y2, Flags [,Superbitmap])
D2, D3, D4 [ A2 ]
struct Layer      *Layer;
struct Layer_Info *LayerInfo;
struct Bit-map     *Bit-map;
LONG               x1,y1,
                   x2,y2;
LONG               Flags;
struct Bitmap    *Superbitmap;]
```

Description: This function creates a layer that is placed in front of all other layers or before all other BACKDROP layers.

Parameters:

LayerInfo: Address of a completely initialized LayerInfo structure

Bit-map: Address of the bit-map in which the layer should be displayed.

x1, y1, x2, y2:
Upper left and lower right corners of the layer.

Flags: Describe layer type.

LayerSimple: Flag to test whether disturbed layer section should be refreshed.

Layersmart: A layer section overlapped by other layers is stored in a memory range allocated by the system for easy refresh. If insufficient memory exists, a Guru Meditation occurs.

Layersuper: Superbitmap support. Superbitmaps are bit-maps that are too large to appear on the screen. This superbitmap must be allocated by you but not displayed. The layer draws a normal bit-map, and the superbitmap is transferred to the layer containing the normal bit-map.

If you draw outside of the layer's border, these lines are transferred directly into the superbitmap. If `SizeLayer()` or `ScrollLayer()` are used, the normal bit-map is copied into the correct position of the superbitmap, and then the new section of the superbitmap is copied back and displayed. When enlarging the layer (`SizeLayer`) the section of the layering coming back is directly displayed.

Backdrop: Operates in conjunction with all other flags (`LAYERSIMPLE`, `LAYERSMART` and `LAYERSUPER` cancel out other flags). A BACKDROP layer stands behind all other layers using `CreateBehindLayer()`, or in front of all of the other BACKDROPS and behind the rest of the layers using `CreateUpFrontLayers()`.

SuperBitmap: Address of the superbitmap that must be presented in conjunction with the `LAYERSUPER` flag.

Layer: Address of a completely initialized layer structure.

Comments: Certain flags must be set in conjunction with certain layer controls. For example, if you want to refresh a `LAYERSIMPLE` layer, the `LAYERREFRESH` flag must be set in `Layer->Flags`. This and the other flags are defined in the include file `graphics/layer.h`. When using the `LAYERSUPER` flag you should also set the `LAYERSMART` flag.

See Also: `CreateBehindLayer()`

FattenLayerInfo — Allocates memory for Layer Info

Syntax:

```
FattenLayerInfo (LayerInfo);
       -156        A0
struct Layer_Info *LayerInfo;
```

Description: This function allocates extra memory for the `Layer_Info` structure. Kickstart Version 1.1 required further layer information. Instead of manually allocating and deallocating memory on each layers library call, `FattenLayerInfo()` predefines the memory allocation. `FattenLayerInfo()` is an old function and should be replaced by `NewLayerInfo()` whenever possible.

Parameter: Layerinfo: Address of the `Layer_Info` structure initialized through `InitLayers()` that should allocate additional memory.

See Also: `InitLayers(),ThinLayerInfo()`

InitLayers — Initializes Layer info structure

Syntax:

```
InitLayers(LayerInfo);
    -30       A0
struct Layer_Info *LayerInfo;
```

Description: This function initializes the given `Layer_Info` structure for the further layer access. After `InitLayers()` `FattenLayerInfo()` must be called. This method is outdated and should be replaced with `NewLayerInfo()` whenever possible.

Parameter: LayerInfo: Address of the `Layer_Info` structure to be initialized.

See Also: `FattenLayerInfo(),ThinLayerInfo()`

NewLayerInfo() — Creates initialized Layer Info structure

Syntax:

```
LayerInfo = NewLayerInfo()
    D0           -144
struct Layer_Info *LayerInfo;
```

Description: This function supplies a completely initialized `Layer_Info` structure. This `Layer_Info` structure remains unlocked after `NewLayerInfo()`.

Parameter: None

Result: LayerInfo: Address of the completely initialized `Layer_Info` structure. When `LayerInfo` = zero, not enough memory could be allocated from `NewLayerInfo()` for the `Layer_Info` structure.

Comment: Through the `Layer_Info` structure the layers created by means of `CreateBehindLayer()` or `CreateUpFrontLayer()` are added to a linked list.

Structures:

```
Offsets  Structure
-------  --------
         struct Layer_Info <graphics/layers.h>
         {
0x00  0        struct Layer *top_layer;
                   /* Address of the top layer */
0x04  4        struct Layer *check_lp,
0x08  8                     *obs;
                   /* System */
0x0c  12       struct MinList FreeClipRects;
0x18  24       struct SignalSemaphore Lock;
0x46  70       struct List gs_Head;
0x54  84       LONG longreserved;
0x58  88       UWORD Flags;
0x5a  90       BYTE fatten_count;
0x5b  91       BYTE LockLayersCount;
0x5c  92       UWORD LayerInfo_extra_size;
0x5e  94       WORD *bltbuff;
0x62  98       struct LayerInfo_extra *LayerInfo_extra;
         }
        struct Layer <graphics/clip.h>
        {

0x00   0       struct Layer *front,
0x04   4                    *back;
                   /* For changing layers */
0x08   8       struct ClipRect *ClipRect;
                   /* Address of clipping rectangles */
0x0c  12       struct RastPort *rp;
                   /* RastPorts of layers */
0x10  16       struct Rectangle bounds;
                   /* Sizes of layers */
0x18  24       UBYTE reserved[4];
0x1c  28       UWORD priority;
0x1e  30       UWORD Flags;
                   /* Flags variable */
0x20  32       struct Bit-map *Superbitmap;
                   /* For LAYERSUPER layer */
0x22  34       struct ClipRect *SuperClipRect;
                   /* Clipping rectangle for superbitmap */
0x26  38       APTR Window; /* Interface to the Intuition
                               Windows */
0x2a  42       SHORT Scroll_X,
0x2c  44             Scroll_Y;
                   /* See ScrollLayer() */
0x30  48       struct ClipRect *cr,
0x34  52                       *cr2,
0x38  56                       *crnew; 0x3c   60
               struct ClipRect *SuperSaveClipRects;
0x40  64       struct ClipRect *_cliprects;
0x44  68       struct Layer_Info *LayerInfo;
                   /* Address of the LayerInfo structure */
0x48  72       struct SignalSemaphore Lock;
```

```
                        /* See LockLayer() */
0x76  118           UBYTE reserved3[8];
0x7e  126           struct Region *ClipRegion;
0x82  130           struct Region *saveClipRects;
0x86  134           UBYTE reserved2[22];
0x9c  156           struct Region *DamageList;
                        /* see BeginUpdate() */
            }
```

6.4.2 Layer processing

BeginUpdate	**Initializes layer update**

Syntax:

```
Status = BeginUpdate (Layer)
  D0          -78         A0
```

Description: This function ensures that the damage list that contains the sections of the layers that must be redrawn is transferred into the `ClipRect` list. This `ClipRect` list contains all of the sections that must be redrawn. When redrawing, only those sections that need redrawing are redrawn. All other sections are undisturbed. `BeginUpdate()` is usually used in conjunction with the `ClipRects`. That way the old damage list is saved:

```
struct Region *OldDamageList;
..
OldDamage-List = Layer->DamageList;
```

Then you make sure that the region that you have previously processed with `OrRectRegion()`,`AndRectRegion()`, etc. is declared as the damage list:

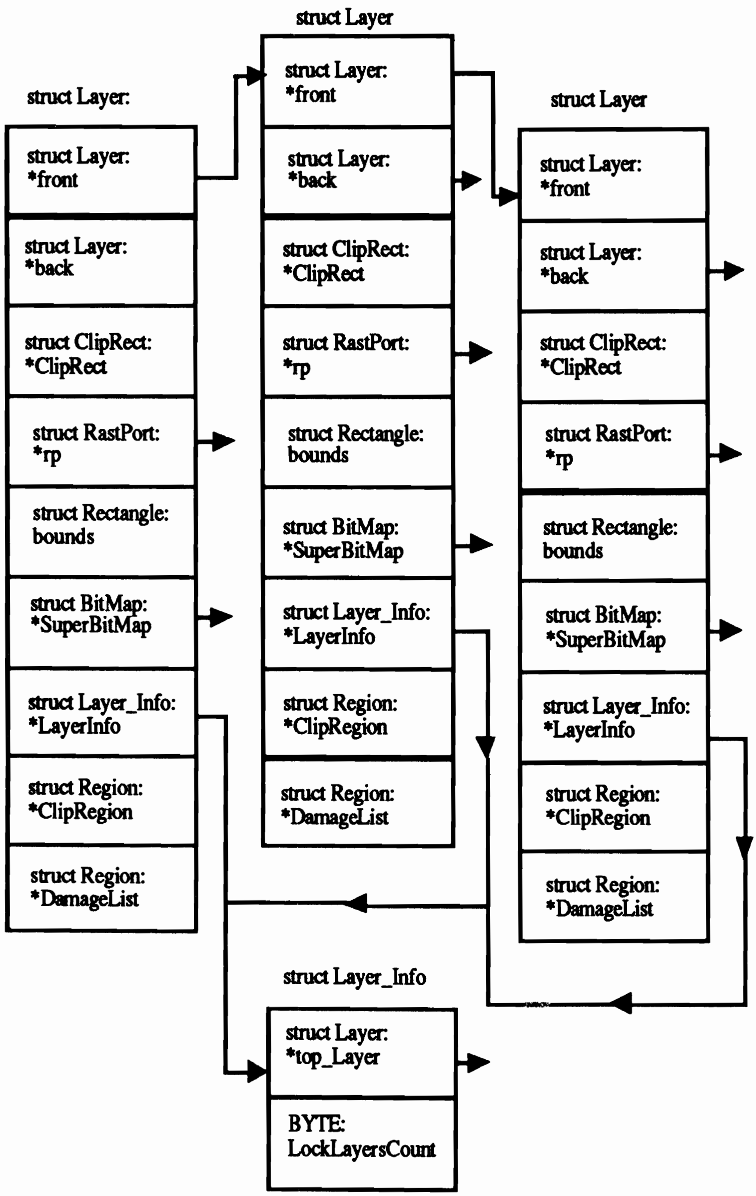
```
struct Region *Region;
..
Layer->DamageList = Region;
```

Now you call `BeginUpdate()`. When you draw in the RastPort of the layer, only the sections of the layer from the drawing operation are met.

Parameter: Layer: Address of the layer whose damage list should function as the `ClipRect` list.

Result: Status: Returns the status of execution. If insufficient memory exists, the status variable returns FALSE.

See Also: `EndUpdate()`


struct Layer:
struct Layer: *front
struct Layer: *back
struct ClipRect: *ClipRect
struct RastPort: *rp
struct Rectangle: bounds
struct BitMap: *SuperBitMap
struct Layer_Info: *LayerInfo
struct Region: *ClipRegion
struct Region: *DamageList
struct Layer
struct Layer: *front
struct Layer: *back
struct ClipRect: *ClipRect
struct RastPort: *rp
struct Rectangle: bounds
struct BitMap: *SuperBitMap
struct Layer_Info: *LayerInfo
struct Region: *ClipRegion
struct Region: *DamageList
struct Layer
struct Layer: *front
struct Layer: *back
struct ClipRect: *ClipRect
struct RastPort: *rp
struct Rectangle: bounds
struct BitMap: *SuperBitMap
struct Layer_Info: *LayerInfo
struct Region: *ClipRegion
struct Region: *DamageList
struct Layer_Info
struct Layer: *top_Layer
BYTE: LockLayersCount

BehindLayer — Places layer in background

Syntax:

```
Status = BehindLayer (Dummy, Layer)
  D0        -54       a0      a1
BOOL          Status;
LONG          Dummy;
struct Layer *Layer;
```

Description: This function places the given layer behind all other existing layers. When sections of a REFRESH layer are visible, the LAYERREFRESH bit for this layer is set.

If the layer in the background is a BACKDROP layer, it is placed behind all of the other BACKDROP layers. Otherwise, the layer is placed behind all other standard layers but in front of any BACKDROP layers.

Parameters:

Dummy: Dummy variable (not used).
Layer: Address layer structure that should be placed in the background.

Result:

Status: Returns TRUE when the process can be executed and FALSE if insufficient memory exists.

See Also: UpFrontLayer()

EndUpdate — Indicates end of update

Syntax:

```
EndUpdate (Layer, Flag);
  -84       a0     d0
struct Layer *Layer;
BOOL          Flag;
```

Description: This function ensures restoration of the ClipRects list after the layer is refreshed.

Parameters:

Layer: Address of the layer that was refreshed.
Flag: Status of the layer's damage list. The value TRUE means that the update was successful. If Flag contains the value FALSE, the old damage list remains available for use as a renewed update cycle (BeginUpdate();... EndUpdate();).

Comments: If you use BeginUpdate() and EndUpdate() to manage your own clippings, make sure that the old damage list is returned to the layer after EndUpdate() (layer damage list = OldDamageList;).

See Also: BeginUpdate()

InstallClipRegion — Adds clipping region

Syntax:

```
OldClipRegion = InstallClipRegion (Layer, Region)
      D0                -174           A0     A1
struct Layer  *Layer;
struct Region *Region;
```

Description: This function clips a specified region in a layer. This takes place after creating a region using `OrRectRegion()` `AndRectRegion()`, etc.

Parameters:

Layer: Address of the layer containing the clipping.
Region: Address of the region that tests the active sections of the layer through clipping rectangles.

Result:

OldClipRegion: Address of the previously installed clipping region.

Comments: Before using `DeleteLayer()` make sure that the actual clipping region is cleared using `InstallClipRegion (layer, null);`. There may not be enough memory available for a normal layer operation (`SizeLayer()`, `MoveLayer()`, etc.). If this is the case, the system uses the memory allocated for the clipping region. When enough memory is present again, the clipping region is restored with a call of a layers library function.

See Also: `BeginUpdate()`, `EndUpdate()`

LockLayer — Denies task access to layer

Syntax:

```
LockLayer (Dummy, Layer);
   -96      A0     A1
LONG            Dummy;
struct Layer *Layer;
```

Description: This function locks layer access away from other tasks. `LockLayer()` waits until the layer is freed from the other tasks (`UnlockLayer()`), and then locks this layer.

Parameters:

Dummy: Dummy variable (not used).
Layer: Address of layer that should be locked.

Comments: After executing `LockLayer()` execution of commands from Intuition like `SizeWindow()`, `MoveWindow()`, etc. are suppressed. Avoid calling Intuition functions while a layer is locked.

See Also: `LockLayerInfo()`, `LockLayers()`

LockLayerInfo — Denies access to Layer Info structure

Syntax:

```
LockLayerInfo (LayerInfo);
      -120        A0
struct Layer_Info *LayerInfo;
```

Description: This function locks access to the `Layer_Info` structure away from other tasks. The layers library waits until the layer is freed from the other tasks. `UnlockLayerInfo()` is called after the execution of each layers function.

`LockLayerInfo()` and `UnLockLayerInfo()` are only needed when the user wants to change a layer structure after it is installed.

Parameter: LayerInfo: Address of the `Layer_Info` structure to be locked.

See Also: `UnLockLayerInfo()`

LockLayers — Denies task access to all layers

Syntax:

```
LockLayers (LayerInfo);
   -108         A0
struct Layer_Info *LayerInfo;
```

Description: This function locks all of the layers in the given `Layer_Info` structure, as well as the given `Layer_info` structure.

Parameter: LayerInfo: Address of the `LayerInfo_Structure` whose layer should be locked.

See Also: `LockLayer()`

MoveLayer — Moves layer on the screen

Syntax:

```
Status = MoveLayer (Dummy, Layer, DeltaX, DeltaY);
  D0        -60      A0     A1      D0      D1
BOOL          Status;
LONG          Dummy;
struct Layer *Layer;
LONG          DeltaX,
              DeltaY;
```

Description: This function moves a layer on the screen. If some sections of other layers are visible, the system creates a damage list for these layers and sets the `LAYERREFRESH` flag.

Parameters:

Dummy: Dummy variable (not used).
Layer: Address of the layer that should be moved (cannot be a BACKDROP layer).

DeltaX, DeltaY:
Number of pixels that the layer should be moved in the X or Y direction.

Comments: If you move the layer to a point outside of the RastPort, a system crash may occur.

MoveLayerInFrontOf — Puts layer in front of another

Syntax:

```
Status = MoveLayerInFrontOf (Layer1, Layer2);
  D0             -168           A0      A1
BOOL          Status;
struct Layer *Layer1,
             *Layer2;
```

Description: This function places `Layer1` in front of `Layer2`. The damage list re-calculates all of the layers and sets the `LAYERREFRESH` flag in the corresponding layers.

Parameters:

Layer1: Address of the layer that should be positioned in front of Layer2.

Layer2: Address of the layer that Layer1 should stand in front of.

ScrollLayer — Changes screen section of a layer

Syntax:

```
ScrollLayer (Dummy, Layer, DeltaX, DeltaY);
     -72       A0     A1     D0      D0
LONG          Dummy;
struct Layer *Layer;
LONG          DeltaX,
              DeltaY;
```

Description: This function moves the contents of a superbitmap layer or normal layer.

Parameters:

Dummy: Dummy variable (not used).

Layer: Address of the layer whose contents should be scrolled.

DeltaX, DeltaY:
Number of pixels that the layer should be moved. Positive delta values move the contents toward the upper left corner of the screen; negative delta values move the contents toward the lower right corner.

SizeLayer — Changes layer size

Syntax:

```
Status = SizeLayer (Dummy, Layer, DeltaX, DeltaY);
  D0         -66        a0     A1      D0      D1
BOOL          Status;
LONG          dummy;
struct Layer *Layer;
LONG          DeltaX,
              DeltaY;
```

Description: This function changes the size of a layer. After sizing a layer, parts of other layers may need to be covered or exposed. The LAYERREFRESH bits are set after `SizeLayer()` to alleviate this. When superbitmap layers are enlarged, the visible sections of the superbitmap are copied into the new regions of the layer.

Parameters:

Dummy: Dummy variable (not used).

Layer: Address of the layer that should be enlarged or reduced.

DeltaX, DeltaY: Number of points that the layer should be enlarged or reduced in the X or Y axis. Positive delta values enlarge the layer; negative delta values reduce the size of the layer.

Result:

Status: Returns FALSE if insufficient memory exists for layer enlargement.

SwapBitsRastPortClipRect — Exchanges RastPort and clipping rectangle bits

Syntax:

```
SwapBitsRastPortClipRect (RastPort, ClipRect);
          -126            A0         A1
struct RastPort *RastPort;
struct ClipRect *ClipRect;
```

Description: This function exchanges the contents of a RastPort with a clipping rectangle. You can then execute graphic operations in the given RastPort that you don't want to execute in the RastPort of the layer (e.g., graphic operations in the background).

Parameters:

RastPort: Address of the RastPort in which the contents of the ClipRects should be copied and whose contents should be copied into the bit-map of the ClipRect.

ClipRect: Address of the ClipRects whose bit-map should be copied in the RastPort.

UpFrontLayer — Moves layer to foreground

Syntax:

```
Status = UpfrontLayer (Dummy, Layer);
  D0        -48          A0     A1
BOOL           Status;
LONG           dummy;
struct Layer *Layer;
```

Description: This function moves a layer to the foreground. When the layer to be moved is a BACKDROP layer, it can only be moved in front of other BACKDROP layers. A BACKDROP layer still remains behind all normal layers.

Parameters:

Dummy: Dummy variable (note used).
Layer: Address of the layer that should be moved to the foreground.

Result:

Status: Returns TRUE if the operation executes without error, and FALSE if not enough memory was available to complete the operation.

See Also: `BehindLayer()`

WhichLayer — Returns layer containing pixel

Syntax:

```
Layer = WhichLayer (LayerInfo, x, y);
  D0      -132           A0      D0 D1
struct Layer      *Layer;
struct Layer_Info *LayerInfo;
SHORT              x,y;
```

Description: This function returns the layer that contains the given pixel of the bit-map.

Parameters:

LayerInfo: Address of the `Layer_Info` structure of the layer that should be searched.
x, y: Screen coordinates of the pixel to be located.

Result:

Layer: Returns the address of the visible layers that contains the given point, or the value 0 if the point cannot be found in any layer.

Structures:

```
Offsets   Structure
-------   --------
          struct ClipRect <graphics/clip.h>
          {
0x00   0       struct ClipRect *Next;
0x04   4       struct ClipRect *prev;
                   /* For linking */
0x08   8       struct Layer *lobs;
0x0c  12       struct Bit-map *Bit-map;
                   /* Bit-map of ClipRect */
```

```
0x10   16      struct Rectangle bounds;
                   /* size of ClipRect */
0x18   24      struct ClipRect *_p1,
0x1c   28                      *_p2;
0x20   32      LONG reserved;
              #ifdef NEWCLIPRECTS_1_1
0x24   36      LONG Flags;
              #endif
          }
```

6.4.3 Releasing layers

DeleteLayer — **Deletes layer**

Syntax:

```
Status = DeleteLayer (Dummy, Layer);
  D0          -90      a0     a1
BOOL          Status;
LONG          Dummy;
struct Layer *Layer;
```

Description: This function removes the specified layer and frees memory allocated for the layer structure by either `CreateUpFrontLayer()` or `CreateBehindLayer()`. In addition the `LAYERREFRESH` flag is set to accommodate the remaining layers.

When using a `LAYERSMART` layer, all of the backup memory is freed. When using a superbitmap layer, the superbitmap remains and makes further graphic manipulations available.

Parameters:

Dummy:	Dummy variable (not used).
Layer:	Address of the layer structure to be freed.

Result:

Status:	Returns TRUE if no error occurred and FALSE if an error occurred.

See Also: `DisposeLayerInfo()`

DisposeLayerInfo — **Frees Layer Info structure**

Syntax:

```
DisposeLayerInfo (LayerInfo)
       -150         a0
struct Layer_Info *LayerInfo;
```

Description: This function frees the memory allocated by `NewLayerInfo()` for the `Layer_Info` structure of the layer. Before calling `DisposeLayerInfo()`, `DeleteLayer()` must be called for each layer of this `Layer_Info` structure.

Parameter: LayerInfo: Address of the `Layer_Info` structure to be freed.

Thin LayerInfo — Releases memory for Layer Info

Syntax:

```
ThinLayerInfo (LayerInfo);
     -162          A0
struct Layer_Info *LayerInfo;
```

Description: This function frees the memory locations allocated for the extra information of a `Layer_Info` structure. This memory was allocated with `FattenLayerInfo()`. `ThinLayerInfo()` is an old function and should be replaced by `DisposeLayerInfo()` whenever possible.

Parameter: LayerInfo: Address of the `LayerInfo` structure whose extra memory should be freed.

See Also: `FattenLayerInfo()`, `InitLayers()`

UnlockLayer — Allows task access to layer

Syntax:

```
UnlockLayer (Layer);
    -102      A0
struct Layer *Layer;
```

Description: This function unlocks the layer to all tasks. The access must have been previously denied using `LockLayer()`.

Parameter: Layer: Address of the layer that should be unlocked.

See Also: `LockLayer()`

UnlockLayers — Allows task access to all layers

Syntax:

```
UnlockLayers (LayerInfo);
    -114          A0
struct Layer_Info *LayerInfo;
```

Description: This function unlocks the layers of the `Layer_Info` structure to all tasks. The access must have been previously denied using `LockLayers()`. `UnlockLayers()` also ensures that the `Layer_Info` structure of the layer is freed (`UnlockLayerInfo()`).

Parameter: LayerInfo: Address of the `Layer_Info` structure whose layers should be unlocked.

See Also: `LockLayers()`

UnlockLayerInfo	**Allows access to Layer Info**

Syntax:

```
UnlockLayerInfo (LayerInfo)
      -128          A0
struct Layer_Info *LayerInfo;
```

Description: This function unlocks the `Layer_Info` structure to all tasks.

Parameter: LayerInfo: Address of the `layer_info` structure to be unlocked.

See Also: `LockLayerInfo()`

6.5 The icon library

The icon library's main function is to serve the programmer and help manipulate the Workbench. It makes functions available for loading and saving icons. In addition, there are useful functions that manipulate `FreeLists`, check the `ToolTypes` array, and copy files.

When you create projects (data files) from a tool (application), the project needs icon information to make it accessible from the Workbench. It is easiest to read an icon that is already on the disk and save it with the same name. For that you can use the functions `GetDiskObject`, `PutDiskObject`, and `FreeDiskObject`.

Icon library functions

6.5.1. Workbench object functions

AllocWBObject	Allocates memory for a WBObject

Syntax:

```
Object = AllocWBObject()
  D0          -66
struct WBObject *Object;
```

Description: This function uses memory for a Workbench object and initializes its FreeList.

Result: Object: Pointer to an initialized WBObject structure.

Exceptions: The function returns a value of zero if the memory could not be allocated.

Comments: Avoid using this function, since it is intended for the internal management of the Workbench. Use the disk object functions instead.

See Also: `AllocEntry, FreeEntry, FreeWBObject`

FreeWBObject	Frees memory allocated for a WBObject

Syntax:

```
FreeWBObject(Object)
    -60        A0
struct WBObject *Object;
```

Description: This function frees the memory that's allocated to the given `WBObject` structure, the structure itself and all of the entries in the structure's FreeList.

Parameter: Object: Pointer to a `WBObject` structure.

Comments: Avoid using this function, since it is intended for the internal management of the Workbench. Use the disk object functions instead.

See Also: `AllocEntry, FreeEntry, AllocWBObject`

GetWBObject	Reads WBObject from disk

Syntax:

```
Object = GetWBObject(name)
  D0          -30      A0
struct WBObject *Object;
UBYTE *name;
```

Description: This function reads a `WBObject` structure from disk. The function inserts `.info` in the filename because it handles the file as an Info file.

Parameter: Name: Pointer to the filename. `.info` is added to the filename.

Result: Object: Pointer to a `WBObject` structure.

Exceptions: The function returns a value of zero if the file could not be loaded. `IoErr` returns the exact error message.

Comments: Avoid using this function, since it is intended for the internal management of the Workbench. Use the disk object functions instead.

See Also: `PutWBObject`

PutWBObject — writes a WBObject in the diskette

Syntax:

```
ok = PutWBObject(Name,Object)
D0        -36     A0    A1
BOOL OK;
UBYTE *Name;
struct WBObject *Object;
```

Description: This function writes a `WBObject` structure to disk. The function inserts `.info` in the filename because it handles the file as an Info file.

Parameters:
- Name: Pointer to the filename. `.info` is added to the filename
- Object: Pointer to a `WBObject` structure

Result: OK: Returns FALSE if the `WBObject` structure cannot be written to disk. `IoErr` returns the exact error message.

Comments: Avoid using this function, since it is intended for the internal management of the Workbench. Use the disk object functions instead.

See Also: `GetWBObject`

Structures:

```
struct WBObject <no longer documented from V1.2>
{
0x00   0  struct Node       wo_MasterNode;
0x0E  14  struct Node       wo_Siblings;
0x1C  28  struct Node       wo_SelectNode;
0x2A  42  struct Node       wo_UtilityNode;
0x38  56  struct WBObject  *wo_Parent;
0x3C  60  UBYTE             wo_Flags;
0x3D  61  UBYTE             wo_Type;
0x3E  62  USHORT            wo_UseCount;
0x40  64  char             *wo_Name;
0x44  68  SHORT             wo_NameXOffset;
```

```
0x46  70  SHORT                wo_NameYOffset;
0x48  72  char                *wo_DefaultTool;
0x4C  76  struct DrawerData   *wo_DrawerData;
0x50  80  struct Window       *wo_IconWin;
0x54  84  LONG                 wo_CurrentX;
0x58  88  LONG                 wo_CurrentY;
0x5C  92  char               **wo_ToolTypes;
0x60  96  struct Gadget       *wo_Gadget;
0x64 100  struct FreeList     *wo_FreeList;
0x68 104  char                *wo_ToolWindow;
0x6C 108  LONG                 wo_StackSize;
0x70 112  LONG                 wo_Lock;
0x74 116
}

struct DrawerData <workbench/workbench.h>
{
0x00  0  struct NewWindow    dd_NewWindow;
0x30 48  LONG                dd_CurrentX;
0x34 52  LONG                dd_CurrentY;
0x38 56  /* more extensive under Version 1.1 ! */
};
```

6.5.2 Icon functions

GetIcon — Reads DiskObject structure from disk

Syntax:

```
ok = GetIcon(Name,Icon,Free)
D0     -42    A0   A1   A2
BOOL OK;
UBYTE *Name;
struct DiskObject *Icon;
struct FreeList *Free;
```

Description: This function reads an Info file of the given `DiskObject` structure and uses additional memory as noted in the FreeList. The function inserts `.info` in the filename because it handles the file as an Info file.

Parameters:

Name: Pointer to the filename. `.info` is added to the filename.

Object: Pointer to a `DiskObject` structure.

Free: Pointer to a FreeList.

Result:

OK: Returns FALSE if the `DiskObject` structure cannot be loaded. `IoErr` returns the exact error message.

See Also: `PutIcon`

PutIcon — **Writes DiskObject structure to disk**

Syntax:

```
OK = PutIcon(Name,Icon)
D0     -48    A0   A1
BOOL ok;
UBYTE *Name;
struct DiskObject *Icon;
```

Description: This function writes the given `DiskObject` structure to an Info file. The function inserts `.info` in the filename because it handles the file as an Info file.

Parameters:

Name: Pointer to the filename. `.info` is added to the filename.

Icon: Pointer to a `DiskObject` structure.

Result:

OK: Returns FALSE if the `DiskObject` structure could not be written. `IoErr` returns the exact error message.

See Also: `GetIcon`

Structures:

```
struct DiskObject <workbench/workbench.h>
{
0x00  0  UWORD                 do_Magic;
0x02  2  UWORD                 do_Version;
0x04  4  struct Gadget         do_Gadget;
0x30 48  UWORD                 do_Type;
0x32 50  char                 *do_DefaultTool;
0x36 54  char                **do_ToolTypes;
0x3A 58  LONG                  do_CurrentX;
0x3E 62  LONG                  do_CurrentY;
0x42 66  struct DrawerData    *do_DrawerData;
0x46 70  char                 *do_ToolWindow;
0x4A 74  LONG                  do_StackSize;
0x4E 78
};
WB_DISKMAGIC      0xe310        /* do_Magic */
WB_DISKVERSION    1             /* do_Version */
NO_ICON_POSITION  (0x80000000)  /* do_CurrentX/Y */
```

6.5.3 Disk object functions

FreeDiskObject — **Frees all DiskObject memory**

Syntax:

```
FreeDiskObject(Object)
       -90      A0
struct DiskObject *Object;
```

Description: This routine frees the memory that belongs to the given disk object. The `DiskObject` structure should have been previously added using `GetDiskObject`.

Parameter: Object: Pointer to a `DiskObject` structure.

See Also: `GetDiskObject`

GetDiskObject — Loads DiskObject structure from disk

Syntax:

```
Object = GetDiskObject(Name)
  D0          -78      A0
struct DiskObject *Object;
UBYTE *Name;
```

Description: This function allocates memory for a `DiskObject` structure and reads an Info file. The function inserts `.info` in the filename because it handles the file as an Info file.

Parameters: Name: Pointer to the filename. `.info` is added to the filename.

Result: Object: Pointer to the `DiskObject` structure containing the Info file data.

Exceptions: The function returns zero if memory could not be allocated for the `DiskObject` structure, or if the file could not be loaded. `IoErr` returns the exact error message.

Comments: This function is similar to the `GetIcon` as it uses the allocation of the `DiskObject` structure and the `FreeList`.

See Also: `PutDiskObject`

PutDiskObject — Writes DiskObject structure to disk

Syntax:

```
OK = PutDiskObject(Name,Object)
D0         -84      A0   A1
BOOL OK;
UBYTE *Name;
struct DiskObject *Object;
```

Description: This function writes the given `DiskObject` structure to an Info file. The function inserts `.info` in the filename because it handles the file as an Info file.

Parameters: Name: Pointer to the filename. `.info` is added to the filename.

Object: Pointer to the `DiskObject` structure containing the Info file data.

Result: OK: Returns FALSE if the file could not be written. `IoErr` returns the exact error message.

See Also: `GetDiskObject`

Structures:

```
struct DiskObject <workbench/workbench.h>
{
0x00  0  UWORD                  do_Magic;
0x02  2  UWORD                  do_Version;
0x04  4  struct Gadget          do_Gadget;
0x30 48  UWORD                  do_Type;
0x32 50  char                  *do_DefaultTool;
0x36 54  char                 **do_ToolTypes;
0x3A 58  LONG                   do_CurrentX;
0x3E 62  LONG                   do_CurrentY;
0x42 66  struct DrawerData     *do_DrawerData;
0x46 70  char                  *do_ToolWindow;
0x4A 74  LONG                   do_StackSize;
0x4E 78
};
WB_DISKMAGIC       0xe310        /* do_Magic */
WB_DISKVERSION     1             /* do_Version */
NO_ICON_POSITION   (0x80000000)  /* do_CurrentX/Y */
```

6.5.4 FreeList functions

AddFreeList	**Inserts a memory entry in FreeList**

Syntax:

```
OK = AddFreeList(Free,Mem,Length)
D0        -72      A0   A1   A2
BOOL OK;
struct FreeList *Free;
UBYTE *Mem;
ULONG Length;
```

Description: This function inserts the memory region specified by `Mem` and `Length` into the `FreeList`. Before accessing this function, the memory must have been previously allocated using `AllocMem`.

Parameters:

Free: Pointer to a `FreeList` structure.
Mem: Pointer to the beginning of the memory range.
Length: Length of the memory range in bytes.

Result: OK: Returns FALSE when the memory could not be taken into the `FreeList`.

See Also: `AllocEntry, FreeEntry, FreeFreeList`

FreeFreeList — Frees all FreeList memory

Syntax:

```
FreeFreeList(free)
     -54      A0
struct FreeList *free;
```

Description: This function frees the `FreeList` structure, as well as all memory as noted in the `FreeList`. A `FreeList` is a list whose elements are `MemList` structures.

Parameter: Free: Pointer to a `FreeList` structure.

Comments: If the `FreeList` structure itself is contained in the `FreeList`, it must be noted in the first element of the `FreeList`.

See Also: `AllocEntry`, `FreeEntry`, `AddFreeList`

Structures:

```
struct FreeList <workbench/workbench.h>
{
0x00  0  WORD                fl_NumFree;
0x02  2  struct List         fl_MemList;
0x10 16
};
struct  MemList <exec/memory.h>
{
0x00  0  struct Node       ml_Node;
0x0E 14  UWORD             ml_NumEntries;
0x10 16  struct MemEntry   ml_ME[1];
0x18 24
};
struct  MemEntry <exec/memory.h>
{
0x00 0  union
        {
0x00 0     ULONG   meu_Reqs;  /* memory flags or  */
0x00 0     APTR    meu_Addr;  /* pointer to memory */
        } me_Un;
0x04 4  ULONG   me_Length;
0x08 8
};
Speicherflags (meu_Reqs);
MEMF_PUBLIC  (1<<0)
MEMF_CHIP    (1<<1)
MEMF_FAST    (1<<2)
MEMF_CLEAR   (1<<16)
MEMF_LARGEST (1<<17)
```

6.5.5 Utility functions

BumpRevision **Changes filename to "copy of ..."**

Syntax:

```
Result = BumpRevision(newBuffer,oldName)
  D0          -108       A0       A1
UBYTE *rResult;
UBYTE *newBuffer,oldName;
```

Description: This function creates a copy of a file, appending "Copy of..." to the beginning of the copy's filename.

Parameters:

newBuffer: Pointer to a buffer in which the new name is stored. This can be up to 31 characters in length (i.e., the maximum length of the DOS filename plus one character).

oldName: Pointer to the filename from which the copy should be made.

Result: Result: Pointer to the new buffer.

Exceptions: If the new name is longer than 30 characters, the name is truncated to 30 characters.

Comments: The maximum length of the DOS filename is currently 30 characters. This number may change in later versions of DOS.

Examples:

oldName	newBuffer
"Test"	"copy of Test"
"copy of Test"	"copy 2 of Test"
"copy 2 of Test"	"copy 3 of Test"
"copy 199 of Test"	"copy 200 of Test"
"copy Test"	"copy of copy Test"
"copy 0 of Test"	"copy 1 of Test"

FindToolType **Searches for ToolType entry**

Syntax:

```
Value = FindToolType(toolTypeArray,Name)
  D0        -96            A0      A1
UBYTE *Value;
UBYTE **toolTypeArray,*Name;
```

Description: This function searches the `ToolType` array for a certain entry. The `ToolType` array consists of pointers to strings which are constructed in the following way:

```
VARIABLE=value
```

You get a pointer to the value of the string back, rather than a copy of the value string.

Parameters:

toolTypeArray: Pointer to the ToolType array.

Name: Pointer to the variable name that should be searched in the array. This is found in the region outside of the array and is not a part of it.

Result: Value: Pointer to the value string.

Exceptions: When the variable could not be found, you get a zero back.

Comments: ToolTypes are assigned files (programs) and can be set from the Workbench with the menu point "Info".

Example: The ToolType array contains the following strings:

```
UBYTE *tTA[] =
{
   "FILETYPE=text",
   "TEMPDIR=:t"
}
```

Then the following function calls result:

```
FindToolType(tTA,"FILETYPE") => pointer to "text"
FindToolType(tTA,"TEMPDIR")  => pointer to ":t"
FindToolType(tTA,"MAXSIZE")  => Null
```

See Also: MatchToolType

MatchToolValue — Checks for ToolType value(s)

Syntax:

```
OK = MatchToolValue(typeString,Value)
D0        -102        A0        A1
BOOL  OK;
UBYTE *typeString,*Value;
```

Description: This function determines whether a ToolType variable contains one or more variables. Multiple values can be separated by | characters.

Parameters:

typeString: Pointer to the variable type (see example) below.

Value: Pointer to the value of a ToolType variable (e.g., a variable found by the FindToolType function).

Result: OK: Returns FALSE if the variable does not contain the value being searched for.

See Also: FindToolType

6.6 The graphics library

The graphics library makes graphic commands available for almost any graphic application. In addition to simple lines, pixel setting and circle drawings, this library contains functions for controlling sprites, bobs and vsprites. You'll also find some functions in this library that are located in the layers library as well. The graphics library is opened as follows:

```
"GfxBase = (struct GfxBase) OpenLibrary("graphics.library",0)"
```

The `GfxBase` structure looks like the following:

```
Offset
------
                    struct GfxBase <graphics/gfxbase.h>
                    {
0x00    0                   struct Library LibNode;
0x22   34                   struct View *ActiView;
                                /* view currently presented */
0x26   38                   struct copinit *copinit;
0x2a   42                   long *cia;

0x2e   46                   long *blitter;

0x32   50                   UWORD *LOFlist;
0x36   54                   UWORD *SHFlist;
0x3a   58                   struct bltnode *blthd,
0x3e   62                                  *bltt1;
                                /* List for QBlit() */
0x42   66                   struct bltnode *bsblthd,
0x46   70                                  *bsbltt1;
                                /* List for QBSBlit () */
0x4a   74                   struct Interrupt vbsrv,
                                /* vertical blank server */
0x60   96                                    timsrv,
                                /* time server */
0x76  118                                    bltsrv;
                                /* blitter server */
0x8c  140                   struct List TextFonts;
                                /* System font list */
0x9a  154                   struct TextFont *DefaultFont;
0x9e  158                   UWORD Modes;
0xa0  160                   BYTE VBlank;
0xa1  161                   BYTE Debug;
0xa2  162                   SHORT Beamsync;
0xa4  164                   SHORT system_bplcon0;
0xa6  166                   UBYTE SpriteReserved;
                                /* reserved Sprites */
0xa7  167                   UBYTE bytereserved;
0xa8  168                   USHORT Flags;
0xaa  170                   SHORT BlitLock;
0xac  172                   short BlitNest;
0xae  174                   struct List BlitWaitQ;
                                /* Blitter Wait Queue */
0xbc  188                   struct Task *BltOwner;
```

```
                              /* who's calling OwnBlitter()? */
0xc0  192                  struct List TOF_WaitQ;
                               /* Top Of Frame Wait Queue */
0xce  206                  UWORD DisplayFlags;
                               /* NTSC, PAL, GENLOCK, etc. */
0xd0  208                  struct SimpleSprite **SimpleSprites;
0xd4  212                  UWORD MaxDisplayRow;
0xd6  214                  UWORD MaxDisplayColumn;
0xd8  216                  UWORD NormalDisplayRows;
0xda  218                  UWORD NormalDisplayColumns;
0xdc  220                  UWORD NormalDPMX;
0xde  222                  UWORD NormalDPMY;
0xe0  224                  struct SignalSemaphore
                           *LastChanceMemory;
0xe4  228                  UWORD *LCMPtr;
0xe8  232                  UWORD MicrosPerLine;
0xea  234                  ULONG reserved[2];
0xf2  242    }
```

Graphics library functions

6.6.1 Raster initialization and processing

6.6.2 RastPort drawing functions

6.6.3 RastPort fill functions

6.6.4 ColorMap functions

6.6.5 Blitter functions

6.6.6 Copper functions

6.6.1 Raster initialization and processing

AllocRaster — **Allocates memory for bit-plane**

Syntax:

```
Memory = AllocRaster (Width,Height)
   D0        -492         D0    D1
PLANEPTR Memory;
SHORT    Width,Height;
```

Description: This function allocates as much memory as needed to accommodate a bit-plane `Width * Height` pixels in size.

Parameters:

Width: Bit-plane width in pixels.
Height: Bit-plane height in pixels.

Result:

Memory: Returns the starting address of the memory to be allocated. When the requested number of bytes is reserved, this function returns a value of zero.

See Also: `FreeRaster()`

FreeRaster — **Frees allocated bit-plane memory**

Syntax:

```
FreeRaster (Memory, Width, Height)
    -498      A0      D0     D1
PLANEPTR BitPlane;
SHORT    Width, Height;
```

Description: This function releases the memory previously allocated for a bit-plane using `AllocRaster()`. This makes the memory available to any part of the system.

Parameters:

Memory:	Address of the memory to be freed.
Width:	Bit-plane width in pixels.
Height:	Bit-plane height in pixels.

Comments: You must give the same values for `Height` and `Width` as you did in `AllocRaster`. If these values are different than the ones used to allocate memory, too little or too much memory is freed.

See Also: `AllocRaster()`

InitBitMap — Initializes BitMap structure

Syntax:

```
InitBitMap (BitMap, Depth, Width, Height)
   -390      A0      D0     D1     D2
struct BitMap *BitMap;
BYTE           Depth,
SHORT          Width,
```

Description: This function initializes a `BitMap` structure.

Parameters:

Bit-map:	Address of the `BitMap` structure to be initialized.
Depth:	Number of bit-planes the bit-map should contain.
Width:	Bit-map width in pixels.
Height:	Bit-map height in lines.

Comments: After the initialization of the bit-map structure using `InitBitMap()`, you must assign the address of the bit-planes in the structure `(BitMap.Planes[i] = memory;)`. The size of each bit-plane corresponds to that of the bit-map.

See Also: `AllocRaster()`

InitRastPort — Initializes RastPort structure

Syntax:

```
InitRastPort (RastPort)
     -198       A1
struct RastPort *RastPort;
```

Description: This function initializes a `RastPort` structure.

Parameter:

RastPort:	Address of the `RastPort` structure to be initialized.

Comments: The `RastPort` structure contains the actual character colors, the actual drawing mode, and more.

After initialization the drawing mode defaults to `JAM2`, and the variables `Mask`, `FgPen`, `AOlPen` and `LinePtrn` contain a value of –1. All of the other variables of the `RastPort` structure are set to zero. So you must make the bit-map the RastPort's "easel" `(RastPort.BitMap = BitMap)`. After that the RastPort is ready for the graphic commands.

SetAPen — Sets foreground pen

Syntax:

```
SetAPen (RastPort, ColorPen)
  -342      A1        D0
struct RastPort *RastPort;
SHORT            ColorPen;
```

Description: This function specifies the foreground pen color.

Parameters:

RastPort: Address of the RastPort whose foreground pen should be changed.

ColorPen: Number of the color register used when drawing with the foreground pen.

Comments: Although only 32 color registers normally exist, the `ColorPen` parameter can accept a value higher than 32. This occurs mainly when the RastPort is the RastPort of an `EXTRA_HALFWIDTH` or `HAM` ViewPort.

See Also: `SetBPen()`, `SetOPen()`

SetBPen — determination of the background pen

Syntax:

```
SetBPen (RastPort, ColorPen)
   -348     A1      D0
struct RastPort *RastPort;
SHORT            ColorPen;
```

Description: This function specifies the background pen color.

Parameters:

RastPort: Address of the RastPort whose background pen should be changed.

ColorPen: Number of the color register used when drawing with the background pen.

Comments: The background pen can be found in `JAM2` mode and any combinations using `JAM2` mode (`JAM2 | COMPLEMENT` and `JAM2 | INVERSVID`) in conjunction with text or area fills. The points not actually set in `JAM1` mode are the color of the background pen in `JAM2` drawing mode (`JAM2 | INVERSVID`).

See Also: `SetDrMd()`, `SetAPen()`

SetDrMd — Sets drawing mode

Syntax:

```
SetDrMd (RastPort, DrawMode)
  -354      A1       D0
struct RastPort *RastPort;
SHORT            DrawMode;
```

Description:

This function sets the drawing mode of a RastPort. The drawing mode dictates the behavior of a pixel, line, etc., in conjunction with pixels already placed in the RastPort. In JAM1 mode the point appears directly in the RastPort. Text output requires special attention. For example, free areas inside of the letter O show through the rest of the letter.

The JAM2 mode is different. There the color of the BPen is assigned to all of the points not set by letters (e.g., the inside of the letter O). In addition, the BPen color covers any 0 bits (e.g., a single color fill pattern). This means that the Blitter can only copy rectangular areas, which can contain these areas. The empty area can be made any color you want.

The following drawing modes function only in conjunction with the JAM1 or JAM2 mode:

COMPLEMENT mode pixels are run through XOR before you set them. If you use JAM2 mode in conjunction with COMPLEMENT mode, the APen and BPen exchange roles.

The INVERSVID mode often occurs in conjunction with the Text() function. This mode allows the display of inverse video characters or graphics. When you use JAM2 mode along with INVERSVID, the system behaves as if you are using JAM2 mode only, except that the APen and BPen switch roles.

Parameters:

RastPort:	RastPort in which the drawing mode should be changed.
DrawMode:	Selected drawing mode.

Comments:

DrawMode can have the values JAM1, JAM2, COMPLEMENT, and INVERSVID. These symbols are defined in the include file graphics/rastport.h.

SetDrPt — Sets drawing pattern

Syntax:

```
SetDrPt (RastPort, Pattern)
(Macro)
struct RastPort *RastPort;
UWORD            Pattern;
```

Description:

This function determines the line pattern.

Parameters: RastPort: RastPort in which the line pattern should be changed.
Pattern: New line pattern. Each set bit in this word represents a set pixel in the line.

Comments: This line pattern is only 16 points wide. Any lines wider than 16 pixels repeat the pattern.

SetRast — Sets RastPort colors

Syntax:

```
SetRast (RastPort, ColorPen)
-234       A1        D0
struct RastPort *RastPort;
UBYTE            ColorPen;
```

Description: This function assigns one color to all the pixels in the RastPort.

Parameters: RastPort: Address of the RastPort to be colored in.
ColorPen: Number of the color register in whose color the RastPort should be colored.

See Also: `SetAPen(),SetBPen()`

SetWrMsk — Sets bit-planes for writing

Syntax:

```
SetWrMsk (RastPort, Mask)
(Macro)
struct RastPort *RastPort;
UBYTE            Mask;
```

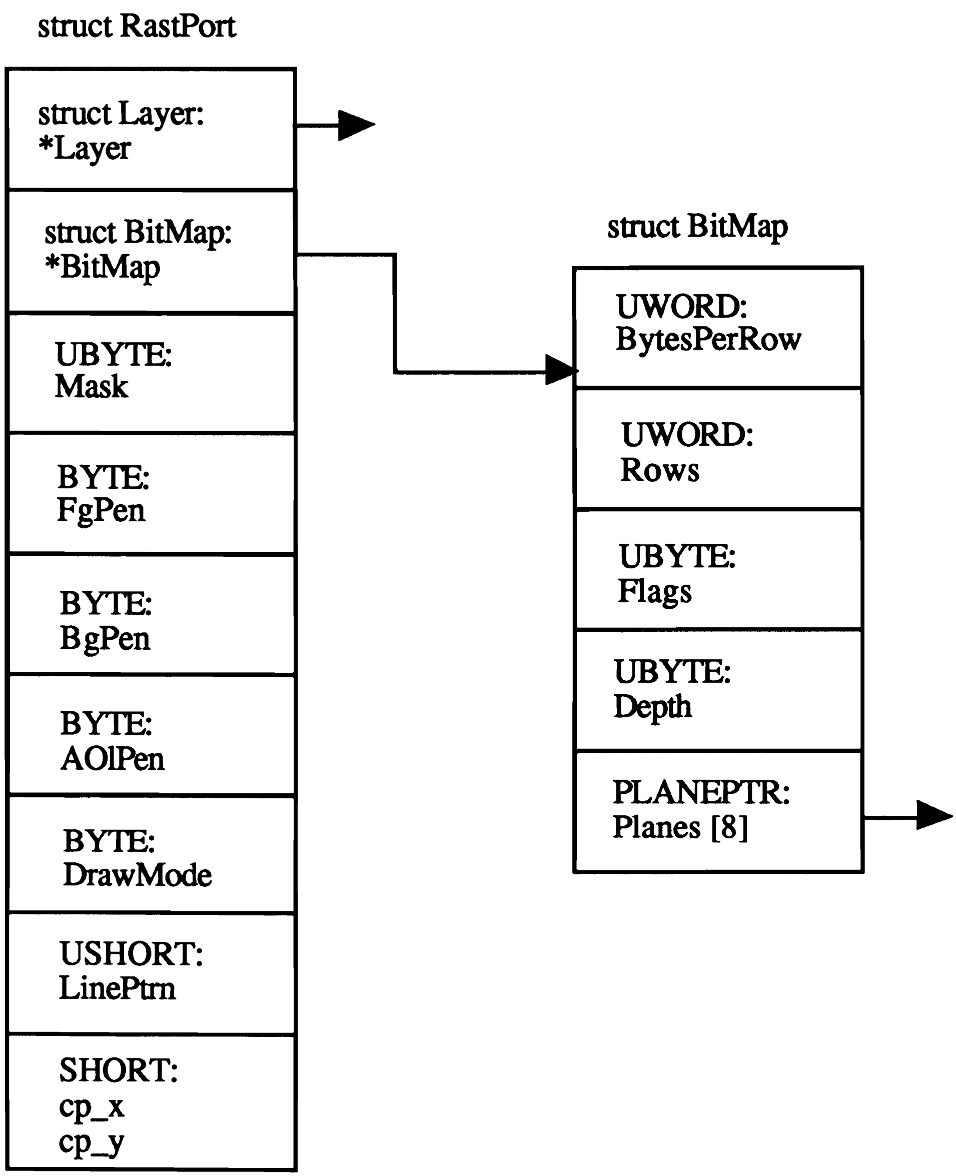
Description: This function determines which bit-planes of a bit-map can be written to (e.g., pixel setting).

Parameters: RastPort: RastPort whose write mask should be changed.
Mask: Bit mask of the bit-planes that can be addressed. Bit 0 represents the bit-plane `BitMap.Planes[0]`, bit 1 `BitMap.Planes[1]`, etc.

Structures:

```
Offset    Structures
------    ----------
          struct BitMap <graphics/gfx.h>
          {
0x00  0       UWORD BytesPerRow;
                 /* Width in bytes */
0x02  2       UWORD Rows;
                 /* Height in lines */
0x04  4       UBYTE Flags;
0x05  5       UBYTE Depth;
                 /* Number of bit-planes */
0x06  6       UWORD pad;
0x08  8       PLANEPTR Planes[8];
                 /* Address of bit-plane */
0x10  16   }
```

```
                struct RastPort <graphics/rastport.h>
                {
0x00      0         struct Layer *Layer;
                        /* Address of layer */
0x04      4         struct BitMap *BitMap;
                        /* Address of bit-map;
0x08      8         USHORT *AreaPtrn;
                        /* Address of fill pattern */
0x0c     12         struct TmpRas *TmpRas;
                        /* Address of temporary raster */
0x10     16         struct AreaInfo *AreaInfo;
                        /* Address of AreaInfo structure */
0x14     20         struct GelsInfo *GelsInfo;
                        /* Address of GelsInfo structure */
0x18     24         UBYTE Mask;
                        /* Write mask */
0x19     25         BYTE FgPen;
                        /* APen */
0x1a     26         BYTE BgPen;
                        /* BPen */
0x1b     27         BYTE AOlPen;
                        /* OPen */
0x1c     28         BYTE DrawMode;
                        /* Drawing mode */
0x1d     29         BYTE AreaPtSz;
                        /* Height of fill pattern */
0x1e     30         BYTE linpatcnt;
0x1f     31         BYTE dummy;
0x20     32         USHORT Flags;
0x22     34         USHORT LinePtrn;
                        /* Line pattern */
0x24     36         SHORT cp_x,
0x26     38               cp_y;
                        /* Graphic cursor position */
0x28     40         UBYTE minterms[8];
0x30     48         SHORT PenWidth;
0x32     50         SHORT PenHeight;
0x34     52         struct TextFont *Font;
                        /* Actual Font */
0x38     56         UBYTE AlgoStyle;
                        /* SoftStyle */
0x39     57         UBYTE TxFlags;
0x3a     58         UWORD TxHeight;
0x3c     60         UWORD TxHeight;
0x3e     62         UWORD TxBaseline;
0x40     64         WORD TxSpacing;
0x42     66         APTR *RP_User;
                        /* User extension */
0x46     70         ULONG longreserved[2];
                   #ifndef GFX_RASTPORT_1_2
0x4e     78         UWORD wordreserved[7];
0x5c     92         UBYTE reserved[8];
                   #endif;
0x64    100 }
```


struct RastPort
struct Layer:
*Layer
struct BitMap:
*BitMap
UBYTE:
Mask
BYTE:
FgPen
BYTE:
BgPen
BYTE:
AOlPen
BYTE:
DrawMode
USHORT:
LinePtrn
SHORT:
cp_x
cp_y
struct BitMap
UWORD:
BytesPerRow
UWORD:
Rows
UBYTE:
Flags
UBYTE:
Depth
PLANEPTR:
Planes [8]

6.6.2 RastPort drawing functions

ClearEOL — **Clears line up to current cursor position**

Syntax:

```
ClearEOL (RastPort)
   -42       A1
struct RastPort *RastPort;
```

Description: This function clears a text line displayed using the RastPort character set, up to the current graphic cursor position. This position can be determined using the `Move()` function. The clearing occurs in such a way that all of the points that lie outside the graphic cursor position are erased, or (in `JAM2` drawing mode) changed to the color of the `BPen`.

Parameter: RastPort: RastPort in which the line should be deleted.

See Also: `ClearScreen(),Move()`

ClearScreen — **Clears screen at current cursor position**

Syntax:

```
ClearScreen (RastPort)
    -48         A1
struct RastPort *RastPort;
```

Description: This function clears the entire screen starting at the current graphic cursor position. The clearing occurs in such a way that all of the points that lie outside the graphic cursor position are erased, or (in `JAM2` drawing mode) changed to the color of the `BPen`.

Parameter: RastPort: RastPort that should be erased from the graphic cursor position to the end of the screen.

See Also: `ClearEOL()`

Draw — **Draws line to given coordinates**

Syntax:

```
Draw (RastPort, x, y)
-246    A1,    D0, D1
struct RastPort *RastPort;
SHORT            x,y;
```

Description: This function draws a line in the given RastPort, from the current graphic cursor position to the specified coordinates. These coordinates become the new graphic cursor position.

Parameters:

RastPort: RastPort in which the line should be drawn.
x, y: Coordinates of the line's end point.

Comments: This function uses the line pattern set using `SetDrPt()`.

See Also: `Move()`

DrawEllipse — Draws ellipse

Syntax:

```
DrawEllipse (RastPort, XM, YM, Xr, Yr)
    -180        A1      D0 D1  D2  D3
struct RastPort *RastPort;
SHORT            XM,YM;
SHORT            Xr,Yr;
```

Description: This function draws the outline of an ellipse in the specified RastPort.

Parameters:

RastPort:	RastPort in which the ellipse should be drawn.
XM, YM:	Coordinates of the ellipse's midpoint.
Xr, Yr:	X and Y radii of the ellipse.

See Also: `AreaEllipse()`

DrawCircle — Draws circle

Syntax:

```
DrawCircle (RastPort, XM, YM, Radius)
(Macro)
struct RastPort *RastPort;
SHORT            XM, YM;
```

Description: This function draws a circle in the specified RastPort.

Parameters:

RastPort:	RastPort in which the circle should be drawn.
XM,YM:	Coordinates of the circle's midpoint.
Radius:	Radius of the circle.

See Also: `AreaEllipse()`

PolyDraw — Draws polygon

Syntax:

```
PolyDraw (RastPort, Number, PointArray)
  -336      A1       D0       A0
struct RastPort RastPort;
SHORT           Number;
struct tPoint   PointArray[MaxNumber];
```

Description: This function draws polygons in the specified RastPort.

Parameters:

RastPort:	RastPort in which the polygon should be drawn.
Number:	Number of corner points in the polygon.
PointArray:	Coordinates of each corner point that should be connected. The X coordinate is saved in `PointArray [i].x` and the Y coordinate is saved in `PointArray [i].y`.

See Also: `Move(),Draw()`

ReadPixel — Reads pixel color

Syntax:

```
ColorPen = ReadPixel (RastPort, x, y)
    D0       -318        A1     D0 D1
LONG             ColorPen;
struct RastPort *RastPort;
SHORT            x,y;
```

Description: This function determines the color of the pixel at the given X/Y coordinates.

Parameters:

RastPort: Address of the RastPort in which a pixel's color should be tested.

x, y: Coordinates of the pixel to be tested.

Result:

ColorPen: Returns the number of the color register whose color is used by the pixel, or the value -1 if x and y are outside the limits of the RastPort's bit-map.

See Also: `WritePixel()`

ScrollRaster — Moves rectangle within RastPort

Syntax:

```
ScrollRaster (RastPort, DeltaX, DeltaY, x1, y1, x2, y2)
     -396        A1       D0      D1    D2  D3  D4  D5
struct RastPort *RastPort;
SHORT            DeltaX, DeltaY;
SHORT            x1,y1,y2,x2;
```

Comments: This function scrolls the contents of a rectangle within the given RastPort.

Parameters:

RastPort: RastPort in which a rectangle should be scrolled.

DeltaX, DeltaY: Coordinates to which the rectangle should be moved. Positive Delta values move the rectangle toward the upper left corner of the RastPort; negative Delta values move the rectangle in the opposite direction.

x1, y1: Upper left corner of the rectangle to be moved.

x2, y2: Lower right corner of the rectangle to be moved.

Comments: `ScrollRaster()` is not especially fast to avoid screen flickering.

Text — String output

Syntax:

```
Text (RastPort, String, NumCharacters)
 -54      A1       A0          D0
struct RastPort *RastPort;
char             *String;
SHORT             NumCharacters;
```

Description: This function displays character strings at the current graphic cursor position (see `Move()`).

Parameters:

RastPort: Address of the RastPort in which the string should be displayed.

String: Address of the string to be written in the RastPort.

NumCharacters: Number of characters that the string to be displayed contains.

Comments: The `Strlen(String)` function easily calculates the number of characters in the string to be displayed.

See Also: `Move()`, `SetDrMd()`

TextLength — Calculates number of pixels

Syntax:

```
Length = TextLength (RastPort, String, NumCharacters)
   D0        -54        A1        A0        D0
SHORT             Length;
struct RastPort *RastPort;
char             *String;
SHORT             NumCharacters;
```

Description: This function computes the horizontal resolution of the string to be displayed. This resolution is in pixels, not in characters. The `TextLength` function is useful for drawing a rectangle around a text, or for centering a rectangle around a text.

Parameters:

RastPort: RastPort into which the string should be written.

String: Address of the string whose width should be tested.

NumCharacters: Number of characters in the string.

Result:

Length: String width in pixels.

Comments: `TextLength` calculates only the width of the string in pixels—the string itself is not displayed. You must display the string using the `Text()` function.

The number of characters of the string to be selected can be computed using the `Strlen(String)` function.

See Also: `Text(),SetFont()`

WritePixel	**Draws single pixels**

Syntax:

```
Status = WritePixel (RastPort, x, y)
            -324        A1    D0 D1
ULONG           Status;
struct RastPort *RastPort;
SHORT           x,y;
```

Description: This function draws one pixel in the current drawing color (`Apen`) at the specified X/Y coordinates of the current RastPort.

Parameters:
- RastPort: Address of the RastPort in which the pixel should be drawn.
- x, y: Coordinates of the pixel to be drawn.

Result:
- Status: Returns 0 if the procedure was successful, and -1 if the given coordinates are outside of the RastPort.

See Also: `ReadPixel()`

Structure:

```
Offset    Structure
------    --------
          struct tPoint <graphics/gfx.h>
          {
0x00  0        WORD x,
0x02  2             y;
0x04  4    }
```

6.6.3 RastPort fill functions

AreaCircle	**Defines circle in AreaInfo structure**

Syntax:

```
Status = AreaCircle (RastPort,Xm,Ym,Radius)
(Macro)
LONG            Status;
struct RastPort *RastPort;
SHORT           Xm,Ym;
SHORT           Radius;
```

Description: This function defines a circle for filling in the `AreaInfo` structure of the given RastPort.

Parameters:
- RastPort: Address of the RastPort in which the filled circle should be drawn.
- Xm, Ym: Coordinates of the circle's midpoint.
- Radius: Radius of the circle to be drawn.

Result: Status: Returns 0 if the data for the circle was accepted by the `AreaInfo` structure, or -1 if the data was rejected.

Comments: `AreaCircle()` is a macro (`graphics/gfxmacros.h`) that `AreaEllipse()` calls with the same X and Y radius.

See Also: `AreaEnd(),InitArea()`

AreaDraw — Defines polygon pixel in AreaInfo

Syntax:

```
Status = AreaDraw (RastPort, x, y)
  D0       -258         A1    D0 D1
LONG             Status;
struct RastPort *RastPort;
SHORT            x,y;
```

Description: This function defines a pixel for a filled polygon.

Parameters:

RastPort: Address of the RastPort in whose `AreaInfo` structure the new polygon pixel should be placed.

x, y: Coordinates of the pixel.

Result: Status: Returns 0 if the data for the polygon was accepted by the `AreaInfo` structure, or -1 if insufficient memory existed for the new pixel.

See Also: `AreaEnd(),AreaEllipse(),AreaMove(),InitArea()`

AreaEllipse — Defines ellipse in AreaInfo structure

Syntax:

```
Status = AreaEllipse (RastPort, Xm, Ym, Xr, Yr)
  D0       -186            A1     D0  D1  D2  D3
LONG             Status;
struct RastPort *RastPort;
SHORT            Xm,Ym;
SHORT            Xr,Yr;
```

Description: This function defines the data for a filled ellipse in the `AreaInfo` structure of the given RastPort.

Parameters:

RastPort: Address of the RastPort in which the filled ellipse should be drawn.

Xm, Ym: Coordinates of the ellipse's midpoint.

Xr, Yr: X and Y radii of the ellipse to be drawn.

Result: Status: Returns 0 if the data for the ellipse was accepted by the `AreaInfo` structure, or -1 if the data was rejected.

See Also: `AreaEnd(),InitArea()`

AreaEnd — Draws polygons, circles or ellipses

Syntax:
```
Status = AreaEnd (RastPort)
  D0        -264    A1
struct RastPort *RastPort;
```

Description: This function draws and fills a polygon created using `AreaMove()` and `AreaDraw()`. Or it draws the determined ellipse with the help of `AreaEllipse()` and fills the surface with the current fill pattern.

Parameter:

RastPort: Address of the RastPort whose `AreaInfo` structure was created using `AreaMove()`, `AreaDraw()`, `AreaCircle()`, or `AreaEllipse()`.

Result:

Status: Returns status of the `AreaEnd()` function, or -1 if insufficient memory is available for filling the area.

Comments: Because `AreaEnd()` draws and fills the polygon/circle/ellipse, the `TmpRas` structure must be initialized in addition to the `AreaInfo` structure.

See Also: `AreaDraw()`, `AreaMove()`, `AreaEllipse()`, `InitArea()`, `InitTmpRas()`

AreaMove — Defines start of polygon in AreaInfo

Syntax:
```
Status = AreaMove (RastPort, x, y)
  D0        -252      A1    D0 D1
struct RastPort *RastPort;
SHORT           x,y;
```

Description: This function defines the starting point of a new polygon. Previously created polygons are ended using `AreaMove()`.

Parameters:

RastPort: Address of the RastPort in which a new polygon should be drawn.

x, y: Starting coordinates of the new polygon.

Result:

Status: Returns -1 if insufficient memory exists for the operation.

See Also: `AreaDraw()`, `AreaEllipse()`, `AreaEnd()`, `InitArea()`

BNDRYOFF — Disables border

Syntax:
```
BNDRYOFF (RastPort)
(Macro)
struct *RastPort;
```

Description: This macro clears the AREAOUTLINE flag in the given RastPort, suppressing the drawing of borders.

Parameter: RastPort: Address of the RastPort in which the AREAOUTLINE bit should be cleared.

Comments: If you want to draw border lines, set the AREAOUTLINE bit in the Flags variables of the RastPort (AREAOUTLINE is defined in graphics/rastport.h).

Flood — Flood fill

Syntax:

```
Flood (RastPort, Mode, x, y)
-330     A1    ,  D2   D0,D1
struct RastPort *RastPort;
ULONG            Mode;
SHORT            x,y;
```

Description: This function fills enclosed areas.

Parameters:

RastPort: Address of the RastPort in which a continuous surface should be filled.

Mode: Test for the enclosed area. If Mode=0, the surface fills with the current fill pattern in the current color, which is bordered by a border line in the color of AOlPen (SetOPen). When mode is 1 the continuous surface receives a new color that has the color of the pixel in the given coordinate.

x, y: Starting coordinates for fill within the RastPort. The pixel color of this coordinate corresponds to the one when Mode == 1.

Comments: The RastPort in which a surface should be filled by means of Flood() must contain a completely initialized TmpRas structure. Because the Area commands also use the Flood command, a TmpRas structure must be present when you use it.

See Also: InitTmpRas(), SetOPen()

InitArea — Initializes AreaInfo structure

Syntax:

```
InitArea (AreaInfo, Buffer, NumPixels)
  -282      A0       A1       D0
struct AreaInfo *AreaInfo;
APTR             Buffer;
SHORT            NumPixels;
```

Description: This function initializes an AreaInfo structure and makes a RastPort available. This must be done before you can use the Area functions.

When this structure is initialized through this function, you specify the address of the coordinates buffer, which must be `NumPoints * 5` bytes, so that it will accommodate a UBYTE array consisting of the number of corner points used in the polygon. For example, at least 25 bytes must be allocated in the coordinates buffer for a four-point polygon ((4+1*) 5 bytes).

The `AreaEllipse()` function needs two coordinates. One ellipse requires a buffer of at least 15 bytes (2*5 bytes for the ellipse itself plus 1*5 bytes for `AreaEnd()`). When you have initialized the `AreaInfo` structure using `InitArea()`, you must add this with `RastPort.AreaInfo = AreaInfo` in the RastPort.

Parameters:

AreaInfo:	Address of the `AreaInfo` structure to be initialized.
Buffer:	Address of the buffer for storing polygon/ellipse/circle data.
NumPixels:	Number of pixels required in the `AreaInfo` structure.

See Also: `AreaDraw(),AreaEnd(),AreaEllipse(),AreaMove()`

InitTmpRas — Initializes TmpRas structure

Syntax:

```
ITmpRas = InitTmpRas (TmpRas, Buffer, BufferSize)
  D0           -468        A0      A1       D0
struct TmpRas *ITmpRas;
struct TmpRas *TmpRas;
APTR           Buffer;
LONG           BufferSize;
```

Description: This function initializes a `TmpRas` structure. `Flood()` and `Area` functions all use the `TmpRas` structure. This operates in conjunction with the recursive fill algorithm, also used by `Flood()`.

This recursive fill algorithm checks the size of the buffer allocated for the operation. This must contain as many bytes as are found in a rectangular area covering the largest area to be filled. Creating a buffer the size of the RastPort's bit-plane or bit-map.

After invoking `InitTmpRas()` the initialized structure must be added to the RastPort (`RastPort.TmpRas = TmpRas`). Unfortunately, each RastPort must have its own completely initialized `TmpRas` structure. Multiple RastPorts cannot share a single `TmpRas` structure.

Parameters:

TmpRas:	`TmpRas` structure to be initialized.
Buffer:	Address of the memory to be assigned to the `TmpRas` structure.
BufferSize:	Buffer size in bytes.

Result: lTmpRas: Returns the address of the initialized `TmpRas` structure. This address is identical to the address of the structure (`lTmpRas = TmpRas`). The `TmpRas` structure can also be given in the RastPort: `RastPort.TmpRas = InitTmpRas (...)`.

RectFill — Fills a rectangle

Syntax:

```
RectFill (RastPort, x1, y1, x2, y2)
  -306       A1      D0  D1  D2  D3
struct RastPort *RastPort;
SHORT            x1,y1,x2,y2;
```

Description: This function fills a rectangle using the current drawing mode, color pen and fill pattern. As soon as the `AREAOUTLINE` bit is set in the RastPort`.Flag` variables a border line appears around the filled rectangle in the color of the `OPen`.

Parameters:

RastPort: Address of the RastPort in which the rectangle should be drawn.

x1, y1, x2, y2: Upper left (x1, y1) and lower right (x2, y2) coordinates of the rectangle. Make sure that the upper left corner coordinates are above and to the left of the lower right coordinates, or a system crash will occur.

Comments: When working in `COMPLEMENT` mode, all of the bit-planes are rotated in the rectangle, as well as those selected using the `APen`.

See Also: `BNDRYOFF()`

SetAfPt — Sets fill pattern

Syntax:

```
SetAfPt (RastPort, Pattern, NumLines)
(Macro)
struct RastPort *RastPort
UWORD         Pattern[];
SHORT         NumLines;
```

Description: This function defines a fill pattern accessed by the `RectFill()`, `Flood()` and `Area...()` functions. You can specify either a single color fill pattern drawn in the `APen` color in the current drawing mode, or a multicolor fill pattern.

Parameters:

RastPort: Address of the RastPort structure to be given a new fill pattern.

Pattern: Bit pattern of the fill pattern.

NumLines: Height of the fill pattern. Only heights that are powers of 2 (0, 1, 2, 4, 8,...) are allowed. Instead of absolute height, you give the power in base 2 that corresponds to the height. Multicolor fill patterns use an exponent preceded by a minus sign. The fill array must then make a bit pattern available for each bit-plane.

Example: The following call assigns a multicolor fill pattern with a height of 16 pixels to the RastPort:

```
WORD Pattern[NumBitPlanes][16];
..
SetAfPt (&RastPort, Pattern, -4);
```

SetOPen — Defines border color pen

Syntax:

```
SetOPen (RastPort, ColorPen)
(Macro)
struct RastPort *RastPort;
SHORT           ColorPen;
```

Description: This macro specifies the border pen color.

Parameters:

RastPort: Address of the RastPort whose border pen should be changed.

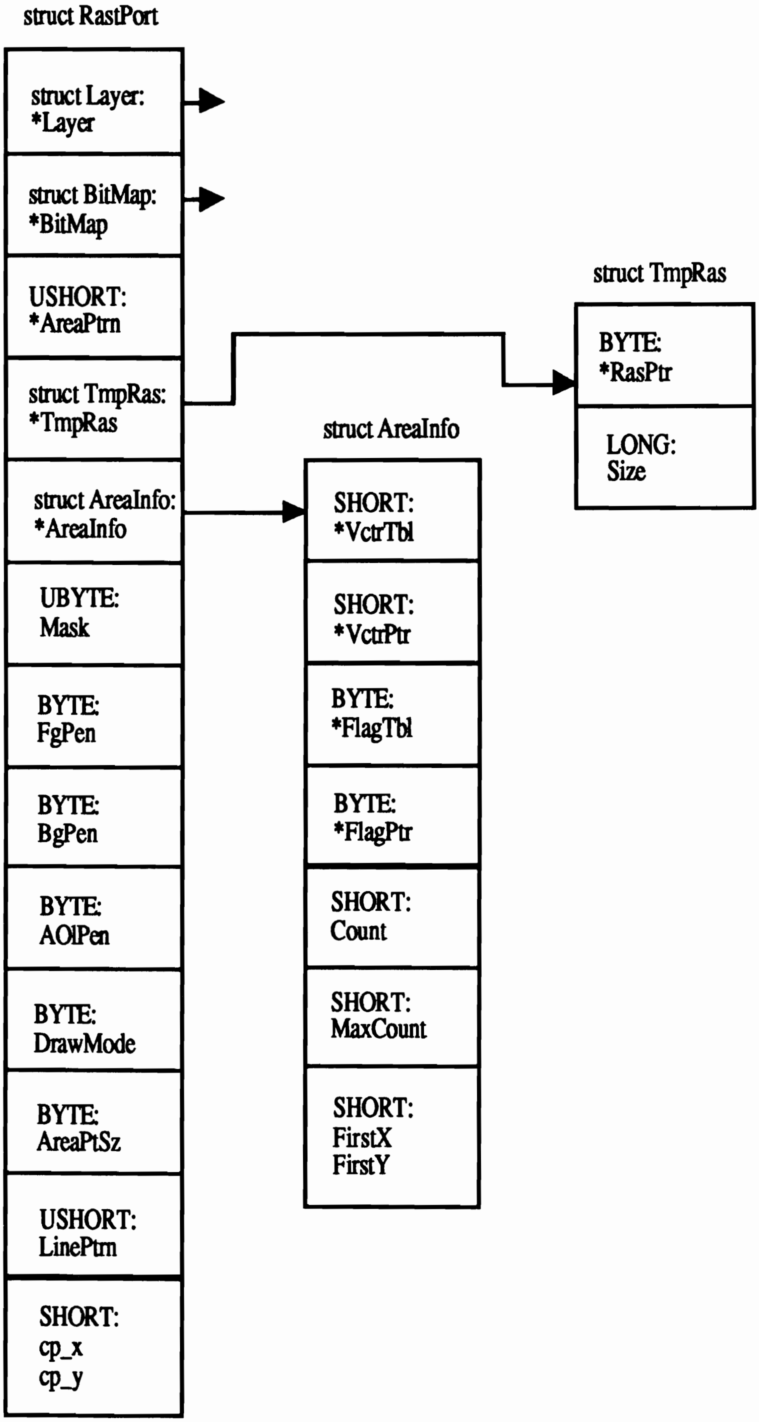
ColorPen: Number of the color register used when drawing with the border pen.

Comments: This macro (found in `graphics/gfxmacros.h`) is used in conjunction with `RectFill()`, `Flood()`, and the `Area()` functions. While it takes an active part in `RectFill()` and `Area...()`, the `OPen` (or `AOlPen`) tests the perimeter of the area to be filled.

See Also: BNDRYOFF()

Structures:

```
Offset   Structures
------   ----------
         struct AreaInfo <graphics/gfx.h>
         {
0x00  0       SHORT *VctrTbl;          /* Vector table */
0x04  4       SHORT *VctrPtr;          /* Next free vector */
0x08  8       BYTE *FlagTbl;
0x0c 12       BYTE *FlagPtr;
0x10 16       SHORT Count;    /* Number of vectors to count */
0x12 18       SHORT MaxCount;
                  /* Maximum numbere of vectors */
0x14 20       SHORT FirstX,
                    FirstY;/* First coordinate (AreaMove()) */
0x18 24   }

         struct TmpRas <graphics/rastport.h>
         {
0x00  0       BYTE *RasPtr;        /* Pointer to raster */
0x04  4       LONG Size;           /* Size of raster */
0x08  8  }
```


struct RastPort
struct Layer:
*Layer
struct BitMap:
*BitMap
USHORT:
*AreaPtrn
struct TmpRas:
*TmpRas
struct AreaInfo:
*AreaInfo
UBYTE:
Mask
BYTE:
FgPen
BYTE:
BgPen
BYTE:
AOlPen
BYTE:
DrawMode
BYTE:
AreaPtSz
USHORT:
LinePtrn
SHORT:
cp_x
cp_y
struct TmpRas
BYTE:
*RasPtr
LONG:
Size
struct AreaInfo
SHORT:
*VctrTbl
SHORT:
*VctrPtr
BYTE:
*FlagTbl
BYTE:
*FlagPtr
SHORT:
Count
SHORT:
MaxCount
SHORT:
FirstX
FirstY

6.6.4 Colormap functions

FreeColorMap **Frees memory from ColorMap structure**

Syntax:

```
FreeColorMap (ColorMap)
    -576         A0
struct ColorMap *ColorMap;
```

Description: This function releases the memory allocated for the `ColorMap` structure. When you allocate memory for a color table using `GetColorMap()`, you must free this memory before ending the program using the `FreeColorMap()` function. This ensures that other applications have access to as much memory as possible.

Parameter: ColorMap: Pointer to the `ColorMap` structure to be freed.

See Also: `GetColorMap()`

GetColorMap **Allocates memory for ColorMap structure**

Syntax:

```
ColorMap = GetColorMap (NumColors)
   D0          -570        D0
struct ColorMap *ColorMap;
LONG             NumColors;
```

Description: This function allocates memory for a `ColorMap` structure, which can then accept `NumColors` color entries.

Parameter: NumColors: The number of color entries actually contained in the `ColorMap` structure.

Result: ColorMap: Returns a pointer to the newly initialized ColorMap structure, given by `(ViewPort.ColorMap = ColorMap)` in the ViewPort. This allows the computation of the Copper lists used by this `ColorMap`.

See Also: `FreeColorMap()`

GetRGB4 **Reads color entry from ColorMap**

Syntax:

```
Color = GetRGB4 (ColorMap, ColorRegister)
   D0     -582     A0            D0
ULONG            Color;
struct ColorMap *ColorMap;
LONG             ColorRegister;
```

Description: This function determines the color combinations used in a single color displayed in the ViewPort. For this you give the number of the color register whose color palette entry you want to read (0-31) in the color register.

Parameters:

ColorMap: Address of the `ColorMap` structure from which you want to read the color entry.

ColorRegister: Number of the color register that you want to read (values range from 0 to 31).

Result:

Color: Word corresponding to the color of the color register being searched. The value in the `Color` variable is coded as follows:

```
Red components   = (Color>>8) & 0xf
Green components = (Color>>4) & 0xf
Blue components  = (Color>>0) & 0xf
```

`Color` returns a value of -1 (0xffff) if the color register number is less than or greater than the 0—31 range.

Comments: You must give the address of a `ColorMap` structure in `GetRGB4()`. If you have not added this to a ViewPort and created the Copper list, the colors of the `ColorMap` are not returned.

See Also: `GetColorMap(), LoadRGB4(),GetRGB4(),SetRGB4CM(), MrgCop(),LoadView()`

LoadRGB4 — Initializes ColorMap

Syntax:

```
LoadRGB4 (ViewPort, ColorPalette, ColorEntries)
  -192      A0        A1             D0
struct ViewPort *ViewPort;
UWORD           ColorPalette[ColorEntries];
SHORT           ColorEntries;
```

Description: This function creates a color palette with different color entries in the `ColorMap` of the specified ViewPort. This adds the colors to the `ColorMap` only; they are not visible in the ViewPort. The sequences `MakeVPort(),MrgCop()` and `LoadView()` must first be called to display the colors in the ViewPort.

Parameters:

ViewPort: Address of the ViewPort whose colors should be changed.

ColorPalette: Color entries written in the corresponding color register after creating the Copper list.

ColorEntries: The number of color entries that should be placed in the ViewPort's color palette.

See Also: `GetRGB4(),SetRGB4()`

SetRGB4 — Changes color entries/color registers

Syntax:

```
SetRGB4 (ViewPort, ColorRegister, Red, Green, Blue)
 -288      A0          D0          D1     D2     D3
struct ViewPort *ViewPort;
SHORT            n;
UBYTE            Red, Green, Blue;
```

Description: This function assigns a new color value to a color register contained in a ViewPort. `SetRGB4()` transfers this color value into the `ColorMap` of the ViewPort, recalculates the Copper list and makes the color change visible immediately.

Parameters:

ViewPort: Address of the ViewPort to be changed.

ColorRegister:
Number of the color register to be changed.

Red, Green, Blue:
Color components. The Amiga creates colors by combining the basic colors red, green and blue in different proportions. Values for each color component range from 0 to 15, resulting in 16^3, or 4096, possible color combinations.

See Also: `LoadRGB4(),GetRGB4(),SetRGB4CM()`

SetRGB4CM — Changes color entry in a ColorMap

Syntax:

```
SetRGB4CM (ColorMap, ColorRegister, Red, Green, Blue)
  -630        A0          D0          D1     D2     D3
struct ColorMap *ColorMap;
SHORT            ColorRegister;
UBYTE            Red, Green, Blue;
```

Description: This function assigns a new color value to a color register contained in a `ColorMap`. Unlike `SetRGB4()`, the Copper list recalculation does not occur, nor does the color change immediately appear on the screen since the new value is not added to the ViewPort. This function is best applied in creating `ColorMaps` that do not need to be displayed immediately.

Parameters:

ColorMap: Address of the `ColorMap` to be changed.

ColorRegister:
Number of the color register to be changed.

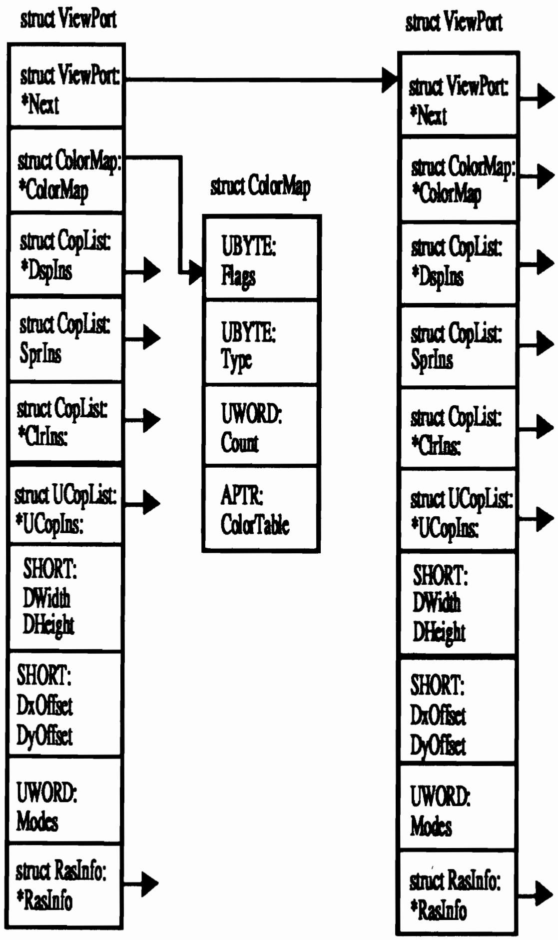
Red, Green, Blue:
Color components. The Amiga creates colors by combining the basic colors red, green and blue in different proportions. Values for each color component

range from 0 to 15, resulting in 16^3, or 4096, possible color combinations.

See Also: `SetRGB4()`

Structures:

```
Offset    Structures
------    ----------
          struct ColorMap <graphics/view.h>
          {
0x00  0         UBYTE Flags;
0x01  1         UBYTE Type;
0x02  2         UWORD Count;
                    /* number of the color entry */
0x04  4         APTR ColorTable;
                    /* Address of the color table */
0x08  8   }
```



6.6.5. Blitter functions

BltBitMap — **Blits rectangle between bit-maps**

Syntax:

```
BitPlanes = BltBitMap (SourceBitMap, x1, y1, DestBitMap,
 D0                    A0            D0  D1      A1
                      x2, y2, Width, Height, Minterm, Mask, Buffer)
                      D2  D3    D4     D5       D6      D7    A2
ULONG          BitPlanes;
struct BitMap *SourceBitMap;
SHORT          x1,y1;
struct BitMap *DestBitMap;
SHORT          x2,y2;
SHORT          Width, Height;
UBYTE          Minterm, Mask;
PLANEPTR       Buffer;
```

Description: This function blits a rectangle from a source bit-map to a rectangle of the same size, either in the same bit-map or a different bit-map. Blitting is more than copying; it can also logically combine the source and destination rectangles.

Parameters:

SourceBitMap:
: Pointer to the bit-map from which the data should be read.

x1, y1:
: Upper left corner coordinates of `SourceBitMap`'s rectangle.

DestBitMap:
: Pointer to the bit-map to which the data should be read or combined.

x2, y2:
: Upper left corner coordinates of `DestBitMap`'s rectangle.

Width, Height:
: Size of the rectangle being blitted.

Minterm:
: Variable which specifies the operation performed by the blitter when creating the destination rectangle. Because `BltBitMap()` can only join two bit-maps together, only the top four bits of this parameter are used:

BC (B and C) = 0x80

$B\overline{C}$ (B and !C) = (B and Not(C)) = 0x40

$\overline{B}C$ (!B and C) = (Not(B) and C) = 0x20

$\overline{BC}$ (!B and !C) = (Not(B) and (Not(C)) = 0x10

B represents SourceBitmap and C represents DestBitmap. Using each of these four Minterms you can copy data or combine the data with information already in the destination rectangle through an AND operator. For example, if you assign 0xC0 to Minterm, the function copies the data from the source rectangle of the source bit-map to the destination rectangle of the destination bit-map. The equation for this operation is (B and C) or (B and !C). You can replace the and with a multiplication symbol (*) and the or with an addition sign (+):

```
(B * C)+(B * !C) (don't forget parenthesis!)
```

After substitution the equation looks like this:

```
B * (C + !C)
```

(C + !C) can be replaced by the number 1 so that only B remains. That's how you can calculate the connection table for all 16 Minterms.

	Mask:	Parameter mask which specifies the accessible bit-planes of the two bit-maps. Only the bit-planes for which the corresponding bit is set are included in the blit (bit 0=BitMap.Planes[0] (first bit-plane), bit 1=BitMap.Planes[2] (second bit-plane), etc.).
	Buffer:	Points to a memory location the size of one line of a bit-plane within a bit-map. If the destination and source bit-maps are identical, the two bit-maps may overlap in some places. The data for the blit is temporarily stored in Buffer. If Buffer is too small, the BitPlanes variable (see **Result** below) returns -1.
Result:	BitPlanes:	Returns the number of bit-planes accessed by the blit, or -1 if Buffer is too small to accommodate the data inserted.
See Also:	ClipBit()	

BltBitMapRastPort — Blits rectangle from bit-map

Syntax:

```
BltBitMapRastPort (SourceBitMap, x1, y1, DestRastPort,
        -606              A0       D0  D1      A1
                     x2, y2, Width, Height, Minterm)
                     D2  D3    D4     D5      D6
struct BitMap    *SourceBitMap;
SHORT             x1,y1;
struct RastPort *DestRastPort;
SHORT             x2,y2;
SHORT             Width, Height;
UBYTE             Minterm;
```

Description: This function blits a rectangle from a source bit-map in a RastPort. Blitting is more than copying; it can also logically combine the source and destination rectangles. The RastPort specifies which bit-planes should be blitted (see also `RastPort.Mask`).

Parameters:

SourceBitMap: Pointer to the bit-map from which the data should be read.

x1, y1: Upper left corner coordinates of `SourceBitMap`'s rectangle.

DestRastPort: Pointer to the RastPort to which the data should be read or combined.

x2, y2: Upper left corner coordinates of `DestRastPort`'s rectangle.

Width, Height: Size of the rectangle being blitted.

Minterm: Variable which specifies the operation performed by the Blitter when creating the destination rectangle (see `BltBitMap()` for more information about `Minterm`).

Mask: Parameter mask which specifies the accessible bit-planes of the two areas. Only the bit-planes for which the corresponding bit is set are included in the blit (bit 0=`BitMap.Planes[0]` (first bit-plane), bit 1=`BitMap.Planes[2]` (second bit-plane), etc.).

See Also: `BltBitMap()`

BltClear — Clears specified memory range

Syntax:

```
BltClear (MemBlk, NumBytes, Flags)
  -300      A1       D0       D1
APTR  Memblk;
ULONG NumBytes;
ULONG Flags;
```

Description: This function clears memory up to the given absolute address using the APTR (Absolute memory PoinTeR, compatible with PLANEPTR [PLANE PoinTeR]). The memory range must be Chip accessible and located in less than 512K.

Parameters:

Memblk: Address of the memory region to clear.

NumBytes: Number of bytes to be cleared.

Flags: Flags which indicate the number of bytes to clear. If bit 1 (=2) of the `Flags` parameter is set, the function reads the top 16 bits of `NumBytes` as the number of rows to be cleared, and the bottom 16 bits as the number of bytes per row of a rectangle to be cleared. If bit 1 of the `Flags` parameter is cleared, `NumBytes` gives the number of bytes to be cleared. Bit 0 of `Flags` indicates whether the program should pause until the Blitter is done clearing memory.

BltMaskBitMapRastPort — Blits through a mask

Syntax:

```
BltMaskBitMapRastPort (SourceBitMap, X1, Y1, DestRastPort,
         -636                  A0       D0  D1       A1
                        X2, Y2, Width, Height, Minterm, BltMask)
                        D2  D3     D4      D5       D6      A2
struct BitMap    *DestBitMap;
SHORT             X1,Y1;
struct RastPort *RastPort;
SHORT             X2,Y2;
SHORT             Width, Height;
UBYTE             Minterm;
APTR              BltMask;
```

Description: This function blits a rectangle from a bit-map into a RastPort, using a mask. This mask is actually a bit-plane, which acts as a pattern for controlling the Blitter operation: Any set pixels in the `BltMask` are allowed through the mask for the blit, while any clear pixels will not penetrate the mask.

Parameters:

SourceBitMap: Pointer to the bit-map from which the data should be read.

x1, y1: Upper left corner coordinates of `SourceBitMap's` rectangle.

DestRastPort: Pointer to the RastPort to which the data should be read or combined.

x2, y2: Upper left corner coordinates of `DestRastPort's` rectangle.

Width, Height: Size of the rectangle being blitted.

Minterm:	Variable which specifies the operation performed by the Blitter when creating the destination rectangle. The `Minterm` variable for this function can have only two values: 0xe0 for a true copy and 0x20 for an inverted source.
BltMask:	Address of the single-plane mask.

See Also: `BltBitMap()`

BltPattern — Fills rectangle using mask

Syntax:

```
BltPattern (RastPort, Mask, x1, y1, x2, y2, NumBytes)
    -312      A1       A0   D0  D1  D2  D3     D4
struct RastPort *RastPort;
APTR             Mask;
SHORT            x1, y1,
                 x2, y2;
SHORT            NumBytes;
```

Description: This function fills a rectangle using the current drawing mode, color pen and fill pattern. The fill pattern is transferred through a mask. This mask is actually a bit-plane, which acts as a template for controlling the fill operation. This mask is the same size as the rectangle being blitted.

Parameters:

RastPort:	Address of the RastPort in which the `BltPattern()` should occur.
Mask:	Address of the single-plane mask.
x1, y1, x2, y2:	Upper left (x1, y1) and lower right (x2, y2) coordinates of the rectangle to be filled. Make sure that the upper left corner coordinates are above and to the left of the lower right coordinates, or a system crash will occur.
NumBytes:	Mask width in bytes. For example, if you want to fill a two-pixel-wide rectangle, enter a 1 here, which indicates that each line of the mask is one byte (eight pixels) wide. Mask width can only be a multiple of 8 (i.e., 8, 16, 24, 32, etc.).

See Also: `BltMaskBitMapRastport()`

BltTemplate — Reads data from packed array

Syntax:

```
BltTemplate (Source, BitPosition, Modulo, RastPort, x,
    -36          A0          D0        D1        A1     D2,
                                             y, Width, Height)
                                             D3   D4     D5

APTR           *Source;
SHORT           BitPosition;
SHORT           Modulo;
struct RastPort *RastPort;
SHORT           x,y;
SHORT           Width,Height;
```

Description:

This function reads a packed segment of data from a rectangular area (Amiga developers call this area a *cookie cut*). Amiga fonts are packed so that they can occupy any width, based on the height assigned to them. The characters are packed bit by bit, without spacing, to optimize memory. With `BltTemplate()`, it allows you to load this data.

Parameters:

Source: Pointer to the start address of the packed data array.

BitPosition: The starting location of the characters within the data array. When the bit pattern begins in the second byte, `BitPosition` contains a value of 16.

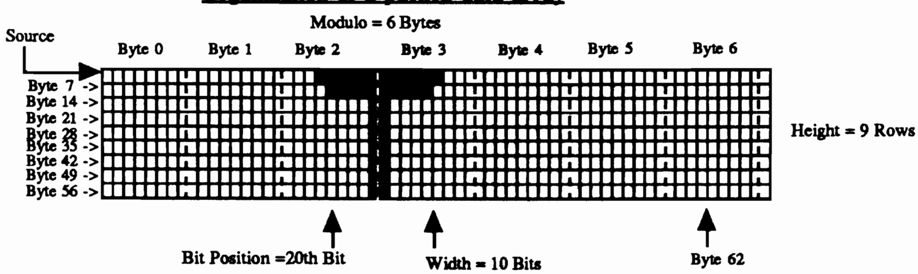
Modulo: The number of bytes that must be added to the current bit position to execute the next line of the data.

RastPort: RastPort to which the read data should be blitted.

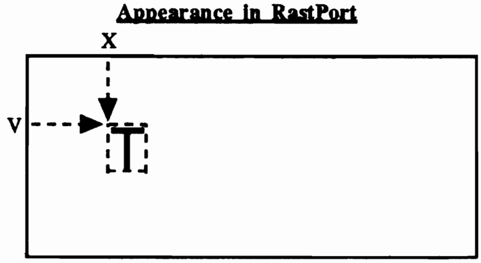
x, y: Upper left coordinates of the rectangle to which the read data should be written.

Width, Height: Character sizes that should be read from the data array.

Organization of a packed data array



Appearance in RastPort



ClipBit — Blits rectangle between RastPorts

Syntax:

```
ClipBit (SourceRastPort, x1, y1, DestRastPort, x2, y2,
 -552           A0       D0  D1      A1       D2, D3,
                                  Width, Height, Minterm)
                                    D4     D5      D6
struct RastPort *SourceRastPort;
SHORT            x1,y1;
struct RastPort *DestRastPort;
SHORT            x2,y2;
SHORT            Width, Height;
UBYTE            Minterm;
```

Description: This function blits a rectangle from a source RastPort to a rectangle of the same size, either in the same RastPort or a different RastPort. The clipping functions only when you use layers and `ClipRects`, or when you use the function within an Intuition screen or window.

Parameters:

SourceRastPort:
: Pointer to the RastPort from which the data should be read.

x1, y1:
: Upper left corner coordinates of `SourceRastPort`'s rectangle.

DestRastPort:
: Pointer to the RastPort to which the data should be read or combined.

x2, y2:
: Upper left corner coordinates of `DestRastPort`'s rectangle.

Width, Height:
: Size of the rectangle being blitted.

Minterm:
: Variable which specifies the operation performed by the Blitter when creating the destination rectangle (see `BltBitMap()` for more information about `Minterm`).

See Also: `BltBitMap()`

DisownBlitter — Releases exclusive Blitter access

Syntax:

```
DisownBlitter()
```

Description: This function releases the Blitter from exclusive access, allowing other programs or tasks to use the Blitter.

See Also: `OwnBlitter()`

OwnBlitter — Reserves Blitter for exclusive use

Syntax:

```
OwnBlitter()
 -456
```

Description: This function locks the Blitter into exclusive access mode (only one task can use the Blitter, excluding all other tasks). `WaitBlit()` should be invoked before Blitter use. This forces the Blitter to wait on executing `OwnBlitter()` until any task currently running through the Blitter is done.

See Also: `DisownBlitter(),WaitBlit()`

QBlit — Inserts BlitNode in Blitter queue

Syntax:

```
QBlit (BlitNode)

 -276
struct blitnode *BlitNode;
```

Description: This function allows Blitter control through routines contained in the `BlitNode` structure of the Blitter job queue.

Parameter: BlitNode: `BlitNode` structure inserted in the Blitter job queue. When your job reaches the head of the queue, you have exclusive access rights to the Blitter.

Comments: The assembly language routine placed in the queue should be written so that it runs in user mode as well as supervisor mode. This type of Blitter control has advantages over `Own/DisownBlitter()` method. When the program/task has executed `OwnBlitter()`, the routine contained in the `BlitNode` structure of the Blitter job queue is executed.

See Also: `QBSBlit()`

QBSBlit — Inserts BlitNode in Blitter queue

Syntax:

```
QBSBlit (BlitNode)
 -294
struct blitnode *BlitNode;
```

Description: This function allows Blitter control through routines contained in the `BlitNode` structure of the Blitter job queue. The routine given in the `BlitNode` structure is called like the `QBlit()` function, except it waits for the electron beam to reach a certain position before executing the job.

This allows screen memory manipulation while the electron beam lies outside of the visible screen area (e.g., the top of the bottom of the screen). The routines are also *beam synchronized*, or synchronized with the electron beam. Because all of these jobs are in a list which is available for all programs and tasks, running problems can occur within one or more tasks.

Parameter: BlitNode: `BlitNode` structure to be inserted.

See Also: `Qblit()`

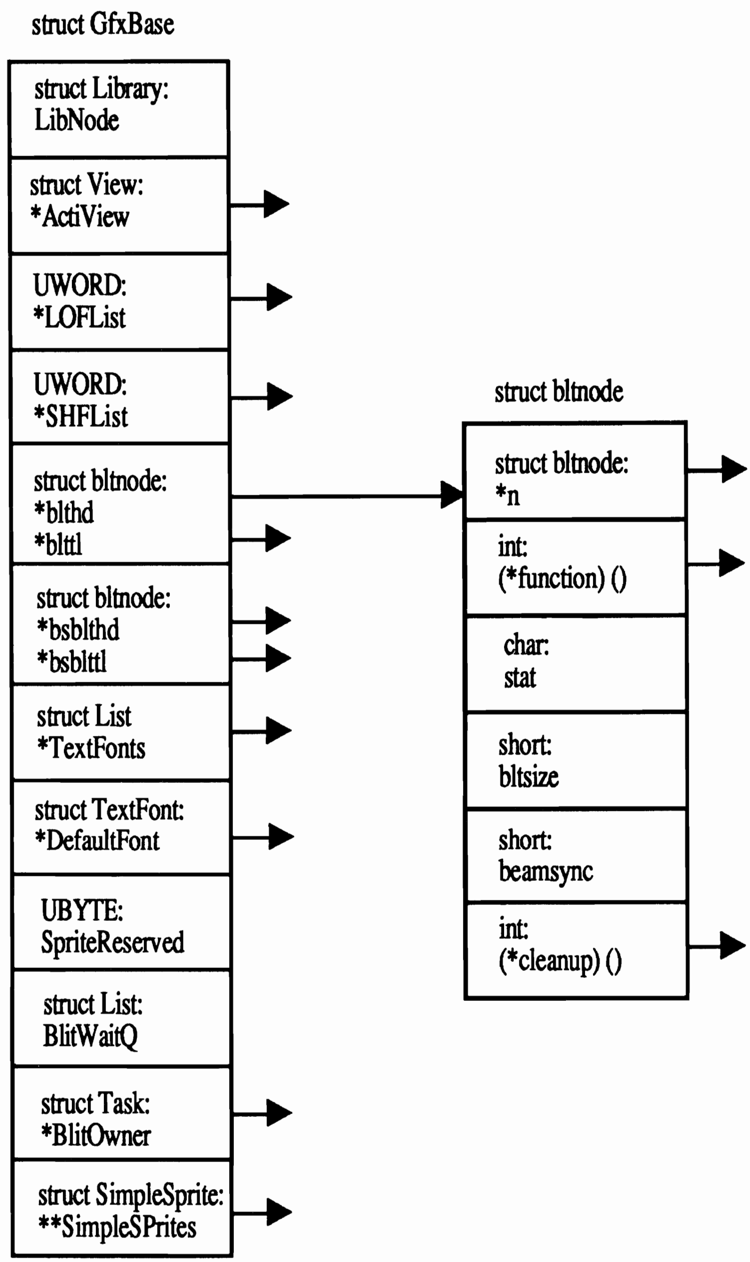
WaitBlit — Waits until end of Blitter access

Syntax: `WaitBlit()`

Description: This command returns to your program or task when the blit, the current blit operation, is completely done. Unfortunately you cannot always get out of `WaitBlit()`. This is because a processor error occurs in Agnus so that `WaitBlit()` comes back although the blit has actually not begun. This can happen especially when the Amiga is running in HIRES with 4 bit-planes.

Structures:

```
Offset    Structures
------    ----------
          struct bltnode <hardware/blit.h>
          {
0x00  0       struct bltnode *n;
                   /* For chaining */
0x04  4       int (*function)();
                   /* Pointer to executing function */
0x08  8       char stat;
                   /* Call cleanup routine */
                   /* ? stat = CLEANUP         */
0x0a 10       short blitsize;
0x0c 12       short beamsync;
                   /* Position of the electron beam */
                   /* for synchronization.          */
0x0e 14       int (*cleanup)();
                   /* Cleanup routine */
0x12 18   }
```


struct GfxBase
struct Library:
LibNode
struct View:
*ActiView
UWORD:
*LOFList
UWORD:
*SHFList
struct bltnode:
*blthd
*bltl
struct bltnode:
*bsblthd
*bsbltl
struct List
*TextFonts
struct TextFont:
*DefaultFont
UBYTE:
SpriteReserved
struct List:
BlitWaitQ
struct Task:
*BlitOwner
struct SimpleSprite:
**SimpleSPrites
struct bltnode
struct bltnode:
*n
int:
(*function) ()
char:
stat
short:
bltsize
short:
beamsync
int:
(*cleanup) ()

6.6.6 Copper functions

CBump — Increments user Copper list command pointer

Syntax:

```
CBump (UCopList)
-366    A1
struct UCopList *UCopList;
```

Description: This function increments the Copper command pointer to the next Copper command's memory location. The programmer usually has little need for this function because the macros `CMove()`, `CWait()` and `CEnd()` do everything necessary to place the desired command in the user Copper list.

Parameter: UcopList: User Copper list into which a command should be entered.

See Also: `CMove(),CWait,UCopListInit`

CEND — Indicates end of user Copper list

Syntax:

```
CEND (UCopList)
(Macro)
struct UCopList *UCopList;
```

Description: This macro from `graphics/gfxmacros.h` marks the end of the user Copper list. Copper lists are saved in the lower 512K of the Amiga, just like most applications that can be processed by the 68000 The `CEnd()` macro informs the Copper that no memory above the Copper list should be executed.

`CEnd()` invokes `CWait(UCopList, 10000, 255)`, which means that before the Copper program can process again it must wait for the electron beam to move to position 255, 10000. This ends the Copper program, since the electron beam cannot move to position 255,10000.

Parameter: UCopList: User Copper list whose end should be marked.

See Also: `CMove(),CWait(),UCopListInit()`

CMove — Writes value in hardware register

Syntax:

```
CMove (UCopList, Register, Value)
-372       A1       D0       D1
struct UCopList *UCopList;
APTR             Register;
SHORT            Value;
```

Description: This function places a command in the user Copper list to write a specific value to a specific hardware register. After this command is placed in the user Copper list, you must increase the Copper list program counter using `CBump()`.

Parameters:

UCopList: User Copper list in which the command should be placed.
Register: Hardware register to which `Value` should be written.
Value: Value that should be placed in `Register`.

Comments: Instead of using `CMove()` and `CBump()` you can use the macro `CMOVE()` (`graphics/gfxmacros.h`). This macro is called using the same parameters as `CMove()`.

You can also invoke `CWait()` to wait for a certain electron beam position, and then change a register (e.g., color register). You may not fill all of the hardware registers with the Copper.

See Also: `UCopListInit(),CWait`

CWait — Waits for electron beam position

Syntax:

```
CWait (UCopList, Y, X)
-378     A1     D0, D1
struct UCopList *UCopList;
SHORT            Y,X;
```

Description: This function places a command in the user Copper list to ensure that the Copper program waits on the processing until the electron beam reaches the specified position. If the electron beam has already passed the specified position, the program continues with the Copper program.

Parameter:

UCopList: User Copper list in which the command should be placed.
Y, X: Y and X coordinates which the electron beam must reach before the Copper commands may continue.

Comments: The `X` parameter should be no greater than 222. Also, instead of the combination `CWait(),CBump()` you can use the macro `CWAIT()` (`graphics/gfxmacros.h`). This is called using the same parameters as `CWait()`.

See Also: CMove(),CWait(),UCopListInit()

FreeCopList **Frees memory in intermediate Copper list**

Syntax:
```
FreeCopList (CopList)
   -546        A0
struct coplist *CopList;
```

Description: This function frees the memory location of a single intermediate Copper list that was created by MakeVPort(). The user does not normally need to call this function because FreeVPortCopList() ensures that all of intermediate Copper lists of a ViewPort are freed.

Parameter: CopList: Pointer to the intermediate Copper list the user wants released.

See Also: FreeVPortCopLists()

FreeCprList **Frees memory of hardware Copper list**

Syntax:
```
FreeCprList (CprList)
    -564     A0
struct cprlist *CprList;
```

Description: This function frees memory that was allocated by the executed Copper list (View.LOFCprList,View.SHFCCprList).

Parameter: CprList: Hardware Copper list to be freed.

See Also: MrgCop()

FreeVPortCopLists **Frees ViewPort Copper lists**

Syntax:
```
FreeVPortCopLists (ViewPort)
      -540           A0
struct ViewPort *ViewPort;
```

Description: This function frees all of the intermediate Copper lists of a ViewPort. The ViewPort can accommodate multiple intermediate lists. One of these lists handles color display, one handles sprite display, etc. FreeVPortCopLists() also frees user Copper lists.

Parameter: ViewPort: ViewPort whose intermediate Copper lists should be freed.

See Also: MakeVPort(),FreeCopList()

InitView	**Initializes View structure**

Syntax:

```
InitView (View)
  -360    A1
struct View *View;
```

Description: This routine clears all variables and initializes a View structure. Then the variables `DxOffset` and `DyOffset`, which control the position of the View on the screen, initialize so that the View's position moves 1/2" from the upper left corner of the screen when the monitor and Preferences are correctly set.

The starting address of the current Copper lists is saved in this View structure. One list is always in use (`View.LOFCprList`) and another comes into service only when interlace mode is invoked (`View.SHFCprList`). When using a resolution mode (HIRES or LACE) remember that a ViewPort arranged within a View can only use the resolution mode that is set in the View.

Parameter: View: View structure to be initialized.

See Also: `MrgCop()`

InitVPort	**Initializes ViewPort structure**

Syntax:

```
InitVPort (ViewPort)
    -204      A0
struct ViewPort *ViewPort;
```

Description: This function clears all variables and initializes a ViewPort structure. The ViewPort is a section of the graphic interface used for actual display. It acts as an interface to the bit-map in which the graphics are saved, and to the `ColorMap` in which the colors are saved.

Intermediate Copper lists must be set (`MakeVPort()`) before the presentation can begin. After that each list must be merged with one another (`MrgCop()`), after which they can be displayed (`LoadView()`).

Parameter ViewPort: ViewPort structure to be initialized.

See Also: `MakeVPort()`, `MrgCop()`, `LoadView()`

LoadView	**Starts View display**

Syntax:

```
LoadView (View)
  -222    A1
struct View *View;
```

Description: This function loads and executes hardware Copper lists. The `MakeVPort()` and `MrgCop()` functions calculate these lists before execution. Most modes, use one list, while interlace mode uses two hardware Copper lists. The starting addresses of these lists appears in the hardware register `cop1lc` (and `cop2lc` if the system is running in interlace mode).

Parameter: View: Address of the View structure for which the hardware list(s) are calculated.

See Also: `InitView(), InitVPort(), MakeVPort(), MrgCop()`

MakeVPort — Calculates Copper lists of a ViewPort

Syntax:

```
MakeVPort (View, ViewPort)
   -216     A0     A1
struct View     *View;
struct ViewPort *ViewPort;
```

Description: This function calculates the intermediate Copper lists for a ViewPort. The ViewPort can accommodate multiple intermediate lists. One of these lists handles color display, one handles sprite display, etc. The individual Copper lists must be merged together using the `MrgCop()` function, after which they are executable using the `LoadView()` function. If you want to display multiple ViewPorts in a View, you must call `MakeVPort()` for each ViewPort.

Parameters:

View: View to which the ViewPort is secondary.

ViewPort: Address of the ViewPort structure from which the intermediate Copper lists should be calculated, based on the `DxOffset` and `DyOffset` values of the View.

See Also: `MrgCop(), LoadView()`

MrgCop — Calculates hardware Copper list

Syntax:

```
MrgCop (View)
 -210    A1
struct View *View;
```

Comments: This function calculates the hardware Copper list based on the intermediate Copper lists of the ViewPort. This ViewPort information is drawn from View.

Parameter: View: Address of the View structure from which the hardware Copper list should be calculated.

See Also: `MakeVPort(), LoadView()`

ScrollVPort — Recalculates Copper lists

Syntax:

```
ScrollVPort (ViewPort)
    -588        A0
struct ViewPort *ViewPort;
```

Description: This function recalculates the Copper list for the ViewPort and adds the result to the hardware Copper list. After you change some variables in the ViewPort structure or in the `RasInfo` structure, the `ScrollVPort()` function offers a simple method of making the changes.

Parameter:

ViewPort: Address of the ViewPort for which the Copper list should be recalculated.

UCopperListInit — Initializes a user Copper list

Syntax:

```
UCopList = UCopperListInit (CopperList, NumberCommands)
                  -594           A0              D0
struct UCopList *UCopList;
struct UCopList *CopperList;
SHORT             NumberCommands;
```

Description: This function re-initializes a previous user Copper list (`Copperlist !=0`) or inserts a new one (`Copperlist == 0`).

Parameters:

CopperList: Pointer to the user Copper list to be re-intialized.

NumberCommands: The number of commands to be placed in the user Copper list.

Result:

UCopList: Returns an initialized `UCopList` structure if `CopperList` equals zero.

Comments: This function can also be called using the `CINIT()` macro (found in `graphics/gfxmacros.h`). The syntax remains the same.

When accessing the user Copper lists you must remember that these are given in the ViewPort in which they should be presented (`ViewPort.UCopIns = UCopList` or `ViewPort.UCopIns = CopperList`). Then you must only calculate the Copper lists (`MakeVPort()`, `MrgCop()`).

See Also: `CBump()`, `CMove()`, `CWait()`

VBeamPos — Reads electron beam position

Syntax:

```
Position = VBeamPos()
              -384
ULONG Position;
```

Description: This function returns the current position of the electron beam that is responsible for drawing the video image. Unfortunately you cannot be 100% certain about the result from this function during multitasking, because multitasking returns slightly inaccurate results. The most accurate result (up to one line) is returned if you run your task at the highest priority.

Result: Position: Vertical position of the electron beam.

WaitBOVP — Waits until ViewPort is displayed

Syntax:

```
WaitBOVP (ViewPort)
 -402        A0
struct ViewPort *ViewPort;
```

Description: This function returns to your program or tasks when the electron beam displayed the last raster line of the given ViewPort (Bottom Of `ViewPort`).

Parameter: ViewPort: Address of the ViewPort waiting for display.

WaitTOF — Waits at top of frame

Syntax:

```
WaitTOF()
   270
```

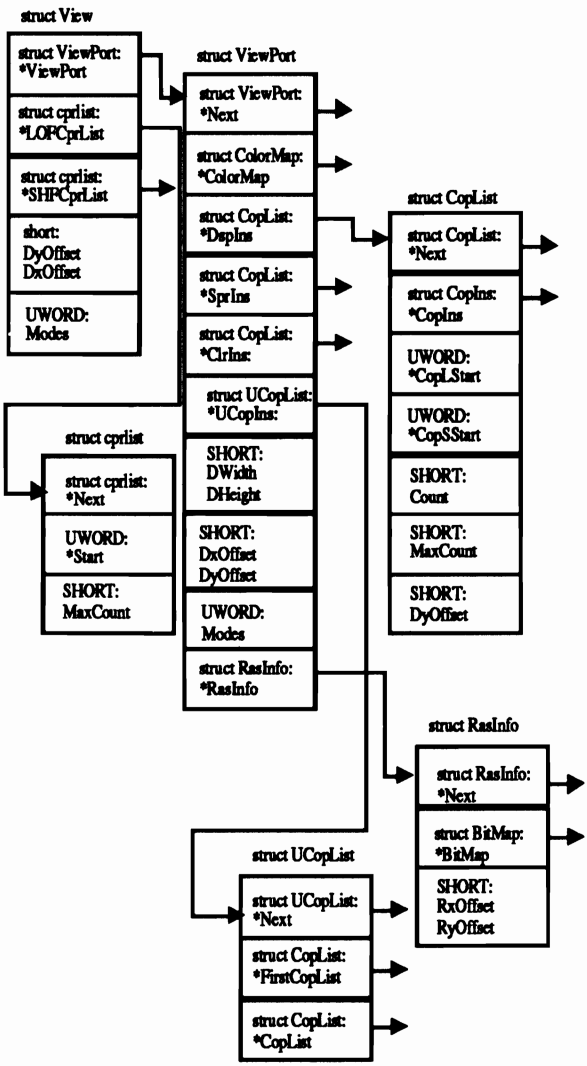
Description: This function waits at the Top Of Frame (TOF) for the electron beam to return. The electron beam travels between the bottom of the screen and the top, displaying nothing. The ViewPort display begins at the first line after the TOF. You can directly change the bottom line of the screen after `WaitTOF()` without an immediately visible change. The task with the highest priority executes immediately after `WaitTOF()`.

Structures:

```
Offset     Structures
------     ----------
           struct UCopList <graphics/copper.h>
           {
0x00   0         struct UCopList *Next;
                     /* For linking */
0x04   4         struct CopList *FirstCopList;
0x08   8         struct CopList *CopList;
0x0c  12   }

           struct CopList  <graphics/copper.h>
           {
0x00   0         struct CopList *Next;
                     /* For linking */
0x04   4         struct CopList *_CopList;
0x08   8         struct ViewPort *_ViewPort;
                     /* Interface to ViewPort */
0x0c  12         struct CopIns *CopIns;
                     /* Copper instructions */
                     /* Linked list */
```

```
0x10  16          struct CopIns *CopPtr;
0x14  20          UWORD *CopLStart;
                      /* Long frame start */
0x18  24          UWORD *CopSStart;
                      /* Short frame start */
                      /* Interlace */
0x1c  28          SHORT Count;
                      /* Number of Copper ins taken  */
0x1e  30          SHORT MaxCount;
                      /* Max. number of Copper ins */
0x20  32          SHORT DyOffset;
                      /* Starting electron beam position */
                      /* of the Copper list.             */
0x22  34    }

            struct cprlist <graphics/copper.h>
            {
0x00   0          struct cprlist *Next;
                      /* for linking */
0x04   4          UWORD *start;
                      /* Start of the Copper list */
0x08   8          SHORT MaxCount;
                      /* number of commands */
0x0a  10    }

            struct ViewPort <graphics/niew.h>
            {
0x00   0          struct ViewPort *Next;
                      /* For linking */
0x04   4          struct ColorMap *ColorMap;
                      /* Address of the ColorMap */
0x08   8          struct CopList *DspIns;
                      /* Display instructions */
0x0c  12          struct CopList *SprIns;
                      /* Sprite instructions */
0x10  16          struct CopList *ClrIns;
                      /* Color instructions */
0x14  20          struct UCopList *UCopIns;
                      /* User instructions */
0x18  24          SHORT DWidth,
0x1a  26                DHeight;
                      /* Size of the ViewPort */
0x1c  28          SHORT DxOffset,
0x1e  30                DyOffset;
                      /* Position of the ViewPorts */
0x20  32          UWORD Modes;
                      /* Display mode */
0x22  34          UBYTE SpritePriorities;
0x23  35          UBYTE reserved;
0x24  36          struct RasInfo *RasInfo;
                      /* Interface to the bit-map */
0x28  40    }

            struct RasInfo <graphics/view.h>
            {
0x00   0          struct RasInfo *Next;
                      /* For DUALPF */
0x04   4          struct BitMap *BitMap;
                      /* Address of the bit-map */
0x08   8          SHORT RxOffset,
0x0a  10                RyOffset;
                      /* Bitmap position, */
```

```
                    /* relative to ViewPort */
0x0c  12  }

          struct View
          {
0x00   0        struct ViewPort *ViewPort;
                    /* Address of the first ViewPort */
0x04   4        struct cprlist *LOFCprList;
                    /* Long frame Copper list */
0x08   8        struct cprlist *SHFCprList;
                    /* Short frame Copper list */
0x0c  12        short DxOffset,
                      DyOffset;
                    /* View position on the monitor */
0x10  16        UWORD Modes;
                    /* Display mode */
0x14  20   }
```



6.6.7 Layer functions

AndRectRegion **ANDs clipping rectangles**

Syntax:

```
AndRectRegion (Region, Rectangle)
      -504       A0        A1
struct Region     *Region;
struct Rectangle *Rectangle;
```

Description: This function combines clipping rectangles in a region into one region. After accessing `AndRectRegion()` only the remaining rectangle in the region is available for drawing.

Parameters:
Region: Region in which the rectangles should be ANDed.
Rectangle: Clipping rectangle that should be ANDed.

See Also: `OrRectRegion()`, `XorRectRegion()`, `OrRegionRegion()`

AndRegionRegion **ANDs two regions**

Syntax:

```
Status = AndRegionRegion (Region1, Region2)
   D0            -624          A0      A1
```

Description: This function removes a section from `Region2` not contained in `Region1` (`Region2 = Region2 AND Region1`).

Parameter:
Region1, Region2:
Regions to be ANDed.

AttemptLockLayerRom **Obtaining access to layer**

Syntax:

```
Status = AttemptLockLayerRom (Layer)
   D0              -654         A5
```

Description: This function tries to gain exclusive access right to a layer. If the layer is unlocked, `AttemptLoclLayerRom` obtains exclusive access.

Parameter:
Layer: Address of the layer to be accessed.

Result:
Status: Returns FALSE if the layer is locked, and TRUE if the layer is unlocked.

ClearRectRegion — Clears a clipping rectangle

Syntax:
```
Status = ClearRectRegion (Region, Rectangle)
  D0          -522         A0       A1
BOOL   Status;
struct Region    *Region;
struct Rectangle *Rectangle;
```

Description: This function clears a clipping rectangle from the specified region.

Parameters:

Region: Address of the region from which the rectangle should be cleared.

Rectangle: Address of the rectangle to be cleared.

See Also: ClearRegion()

ClearRegion — Clears a region

Syntax:
```
ClearRegion (Region)
    -528      A0
struct Region *Region;
```

Description: This function clears all of the clipping rectangles in the specified region, disabling any drawing functions that may follow in a layer.

Parameter:

Region: Region from which the rectangles should be cleared.

See Also: AndRectRegion(), OrRectRegion()

CopySBitMap — Copies superbitmap to layer bit-map

Syntax:
```
CopySBitMap (Layer)
    -450     A0
struct Layer *Layer;
```

Description: This function copies a section of the superbitmap of a LAYERSUPER layer into the bit-map presented by the layer. After calling SyncSBitMap(), you can use the superbitmap graphic operations without worrying about the ClipRects of the layer. After that you should call CopySBitMap().

Parameter:

Layer: Address of the LAYERSUPER layer.

See Also: SyncSBitMap()

DisposeRegion — Frees region memory

Syntax:
```
DisposeRegion (Region)
    -534      A0
struct Region *Region;
```

Description: This function frees the memory for all of the rectangles of the specified region and frees the region memory itself. It also frees the region that was previously allocated using `NewRegion()`.

Parameter: Region: Address of the region to be freed.

See Also: `NewRegion()`

LockLayerRom — Secures access rights to layer

Syntax:

```
LockLayerRom (Layer)
    -432       A5
struct Layer *Layer;
```

Description: This function prevents another program or task from making changes to the specified layer.

Parameter: Layer: Address of the layer that should be locked.

Comments: This routine is identical to the `LockLayer()` function found in the layers library.

See Also: `UnLockLayerRom()`

NewRegion — Allocates new region

Syntax:

```
Region = NewRegion ()
   D0      -516
struct Region *Region;
```

Description: This function allocates and initializes a new region, returning a pointer to the region. The `OrRectRegion` function ensures that clipping rects are accepted in this region.

Only drawing is allowed in these clipping rects. This operates only in conjunction with the layers that use `BeginUpdate()` and `EndUpdate()`.

Result: Region: Pointer to an initialized `Region` structure.

See Also: `OrRectRegion()`, `DisposeRegion()`

OrRectRegion — Inserts clipping rectangle in region

Syntax:

```
Status = OrRectRegion (Region, Rectangle)
             -510          A0        A1
BOOL              Status;
struct Region     *Region;
struct Rectangle *Rectangle;
```

Description: This function adds a clipping rectangle to the specified region. After the `RectRegion` function inserts the rectangle, it can be manipulated using `AndRectRegion()` and `XorRectRegion()`.

Parameters:
Region: Address of the region in which a rectangle should be inserted.
Rectangle: Rectangle structure to be inserted.

Result:
Status: Returns TRUE if enough memory was present for this operation and FALSE if insufficient memory existed.

See Also: `AndRectRegion(),XorRectRegion()`

OrRegionRegion — ORs two regions together

Syntax:

```
Status = OrRegionRegion (Region1, Region2)
  D0           -612
BOOL Status;
struct Region *Region1, *Region2;
```

Description: This function transfers screen sections of `Region1` not contained in `Region2` into `Region2` (`Region2 = Region1 | Region2`).

Parameter:
Region1, Region2: Regions to be `OR`ed.

Result:
Status: Returns TRUE if enough memory was present for this operation and FALSE if insufficient memory existed.

See Also: `NewRegion(),DisposeRegion()`

SyncSBitMap — Copies bit-map to superbitmap

Syntax:

```
SyncSBitMap (Layer)
   -444      A0
struct Layer *Layer;
```

Description: This function copies the presented bit-map to the corresponding location of the superbitmap specified by the `LAYERSUPER` layer. Graphic operations can then be executed in the superbitmap without having to worry about the ClipRects.

Parameter:
Layer: Address of the `LAYERSUPER` layer.

See Also: `CopySBitMap()`

UnlockLayerRom — Free layer

Syntax:

```
UnlockLayerRom (Layer)
     -438       A5
struct Layer *Layer;
```

Description: This function unlocks a layer previously locked into exclusive access by the `LockLayerRom()` function. When the layer locks an Intuition window , `UnLockLayerRom()` frees it so that Intuition can manage window sizing again.

Parameter:

Layer: Address of the layer you want freed.

Warning: The number of `LockLayerRom()` calls must equal the number of `UnLockLayerRom()` calls.

Comments: `LockLayerRom()` and `UnLockLayerRom()` are identical to the layers library functions `LockLayer()` and `UnLockLayer()`.

See Also: `LockLayerRom()`

XorRectRegion — ORs clipping rectangle in region

Syntax:

```
Status = XorRectRegion (Region, Rectangle)
    D0         -558       A0        A1
BOOL                Status;
struct Region     *Region;
struct Rectangle *Rectangle;
```

Description: This function removes a clipping rectangle if contained in a region, and inserts a clipping rectangle if absent from a region.

Parameters:

Region: Address of the region in which the rectangle should be inserted.

Rectangle: Address of the rectangle that should be inserted.

Result:

Status: Returns TRUE if enough memory was present for this operation and FALSE if insufficient memory existed.

See Also: `OrRectRegion(),AndRectRegion()`

XorRegionRegion — Exclusive ORs two regions

Syntax:

```
XorRegionRegion (Region1, Region2)
                  A0       A1
```

Description: This function inserts data from `Region1` into `Region2` if the data differs between the two regions, and clears the data from `Region2` if the data in `Region1` is identical to the data in `Region2`.

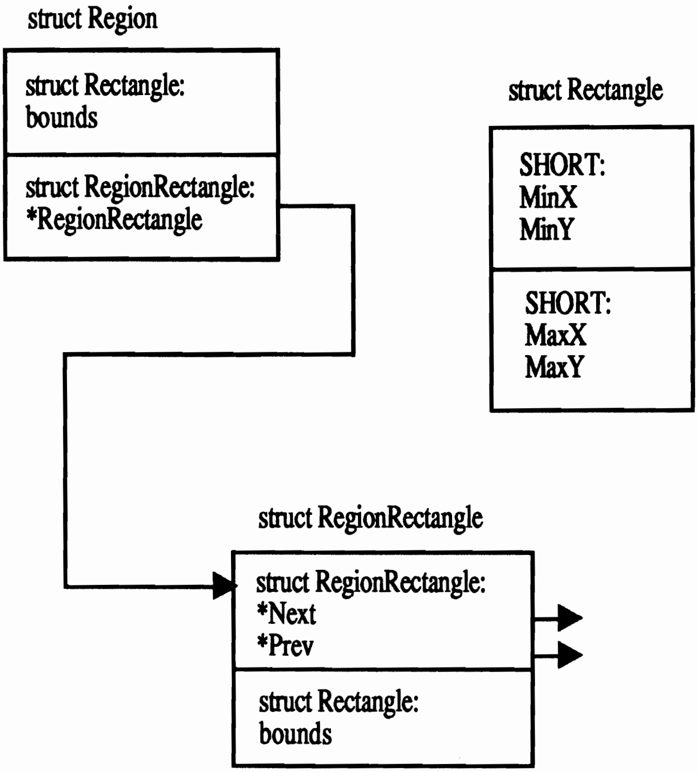
Parameter: Region1, Region2:
Addresses of the two regions that should be exclusive ORed.

Result: Status: Returns TRUE if enough memory was present for this operation and FALSE if insufficient memory existed.

See Also: `OrRegionRegion()`

Structures:

```
Offset    Structure
------    -------
          struct Rectangle <graphics/gfx.h>
          {
0x00   0       SHORT MinX,
0x02   2             MinY;
                   /* Upper left corner */
0x04   4       SHORT MaxX,
0x06   6             MaxY;
                   /* Lower right corner */
0x08   8  }

          struct RegionRectangle <graphics/regions.h>
          {
0x00   0       struct RegionRectangle *Next,
0x04   4                              *Prev;
                   /* For linking */
0x08   8       struct Rectangle bounds;
                   /* size */
0x10  16  }

          struct Region <graphics/region.h>
          {
0x00   0       struct Rectangle bounds;
                   /* Size */
0x08   8       struct RegionRectangle *RegionRectangle;
0x0c  12  }
```


struct Region
struct Rectangle:
bounds
struct RegionRectangle:
*RegionRectangle
struct Rectangle
SHORT:
MinX
MinY
SHORT:
MaxX
MaxY
struct RegionRectangle
struct RegionRectangle:
*Next
*Prev
struct Rectangle:
bounds

6.6.8 Character display

AddFont — **Adds font to system font list**

Syntax:

```
AddFont (TextFont)
 -480      A1
struct TextFont *TextFont;
```

Description: This function makes a font opened by `OpenDiskFont()` the system font. This font can then be opened using the `OpenFont()` functions. Other programs or tasks can access this font without opening the DiskFont library.

Parameter:

TextFont: `TextFont` structure of the font that should be added to the system list.

See Also: `RemFont()`, `OpenFont()`, `OpenDiskFont()`

AskFont — **Describes current font in RastPort**

Syntax:

```
AskFont (RastPort, TextAttr)
   -474      A1       A0
struct RastPort *RastPort;
struct TextAttr *TextAttr;
```

Description: This function determines the text attributes currently in use by the given RastPort's font.

Parameters:

RastPort: Address of the RastPort that should be examined.
TextAttr: Address of the `TextAttr` structure that should be filled with the text attributes of the actual font in the RastPort.

See Also: `SetFont()`

AskSoftStyle — **Reads current font styles**

Syntax:

```
FontStyle = AskSoftStyle (RastPort)
    D0          -84          A1
ULONG           FontStyle;
struct RastPort *RastPort;
```

Description: This function returns the style(s) in use in the current font. Two style types exist:

1. Bit patterns of each character are set in their stylistic form (italic, underlined, etc.).

2. Bit patterns of each character are set for normal display (no italic, etc.).

The `AskSoftStyle()` function tells you the bit pattern as well as reading the font's `SoftStyles`, which precede the specified RastPort. The following styles exist:

```
FSF_UNDERLINED = 1 (Underlined)
FSF_BOLD       = 2 (bold)
FSF_ITALIC     = 4 (italics)
FSF_NORMAL     = 0 (Normal characters)
```

The font style flags are defined in the include file `graphics/text.h`.

Parameter: RastPort: Address of the RastPort structure whose font styles should be read.

Result: FontStyle: Font style flags set by `SetSoftStyle()`.

See Also: `SetSoftStyle()`

CloseFont — Closes font

Syntax:

```
CloseFont (TextFont)
    -78        A1
struct TextFont *TextFont;
```

Description: This function closes a font previously opened by the `OpenFont()` or `OpenDiskFont()` functions. If the font is not entered in the system font list, the function releases the memory allocated for the font.

Parameter: TextFont: Pointer to the `TextFont` structure of the font to be closed.

See Also: `OpenFont(),AddFont()`

OpenFont — Opens font

Syntax:

```
TextFont = OpenFont (TextAttr)
    D0        -72        A0
struct TextFont *TextFont;
struct TextAttr *TextAttr;
```

Description: This function opens a font in the system font list. The `TextAttr` structure describes the desired font as closely as possible. This structure contains the font names (e.g., `TextAttr.ta_Name = "Topaz.Font"`), the size or height (e.g., 9 lines, `TextAttr.ta_YSize = 9`) and the text type that the font should have e.g., `TextAttr.ta_Style = Underlined`).

When a font with the given name is found but the height and text type differ from all of the available fonts, the function opens the font that most closely matches the description given.

Parameter: TextAttr: `TextAttr` structure which describes the font.

Result: TextFont: Pointer to a completely initialized TextFont structure.

See Also: `CloseFont(),AddFont(),RemFont()`

RemFont — Removes font from system font list

Syntax:

```
RemFont (TextFont)
  -486      A1
struct TextFont *TextFont;
```

Description: This function removes a font from the system font list. All programs or tasks that currently have access to the font release their access. If a task tries to access this font using `OpenFont()`, the system denies access.When all tasks indicate that they no longer need the font (using `CloseFont()`), the system frees the memory used by the font.

Parameter: TextFont: Address of the `TextFont` structure that should be removed from the system font list.

See Also: `AddFont()`

SetFont — Set font in RastPort

Syntax:

```
SetFont (RastPort, TextFont)
  -66      A1        A0
struct RastPort *RastPort;
struct TextFont *TextFont;
```

Description: This function controls text output after you have opened a font using `OpenFont()` or `OpenDiskFont()` by assigning the new font to the current RastPort.

Parameters:

RastPort: Address of the RastPort that should receive the new output font.

TextFont: Address of the `TextFont` structure that tests the new output font.

Comments: After initializing the RastPort, the system defaults to the Topaz.font which is also used in the CLI window.

See Also: `OpenFont(),CloseFont()`

SetSoftStyle — Sets a new text type

Syntax:

```
NewTextType = SetSoftStyle (RastPort, TextType, DTextType)
     D0            -90          A1,       D0,        D1
ULONG           NewTextType;
struct RastPort *RastPort;
ULONG           TextType;
ULONG           DTextType;
```

Description: This function assigns a new text style to the font used by `Text()`. Only the text styles whose bits are set in `DTextTypes` can be algorithmically generated.

Parameters:

RastPort: Address of the RastPort whose font should be displayed in a new text style.

TextType: Mask of possible text types (see `AskSoftStyle`).

DTextType: Desired text style in which the given characters should appear through `Text()`. The font style flags (see `AskSoftStyle()`) specify the desired text style.

Result:

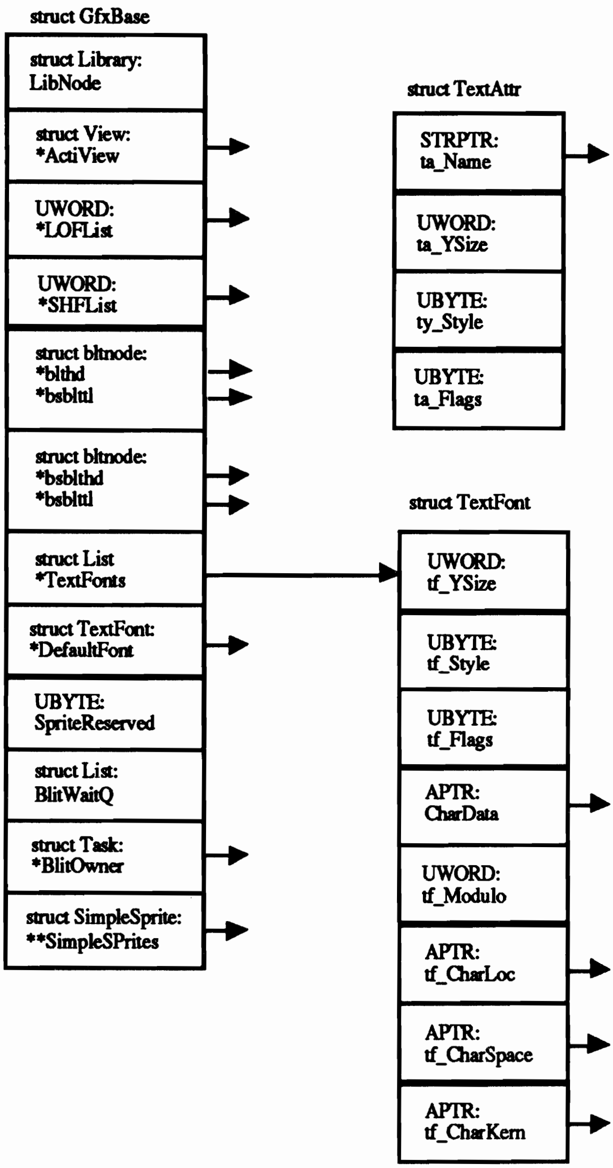
NewTextType: Text types set by `SetSoftStyle()`.

See Also: `AskSoftStyle()`

Structures:

```
Offsets  Structures
-------  ----------
         struct TextAttr <graphics/text.h>
         {
0x00   0        STRPTR ta_Name;
                    /* Font name */
0x04   4        UWORD ta_YSize;
                    /* Height */
0x06   6        UBYTE ta_Style;
                    /* Text style */
0x07   7        UBYTE ta_Flags;
                    /* Font preference flags */
0x08   8  }

         struct TextFont <graphics/text.h>
         {
0x00   0        struct Message tf_Message;
0x14  20        UWORD tf_YSize;
                    /* Height */
0x16  22        UBYTE tf_Style;
                    /* Bit pattern text style */
0x17  23        UBYTE tf_Flags;
                    /* Font preference flags */
0x18  24        UWORD tf_XSize;
0x1a  26        UWORD tf_Baseline;
0x1c  28        UWORD tf_BoldSmear;
0x1e  30        UWORD tf_Accessors;
                    /* number of OpenFont() calls */
0x20  32      UBYTE tf_LoChar; /* Smallest defined ASCII Code */
0x21  33       UBYTE tf_HiChar; /* Largest defined ASCII Code */
```

```
Code 0x22  34       APTR tf_CharData;/* Packed bit pattern
                                        address*/
0x26  38       UWORD tf_Modulo; /* Packed data array module */
0x28  40       APTR tf_ChatLoc;
                     /* Bit pos. addresses of each character */
0x2c  44       APTR tf_CharSpace;
                   /* Character container size */
0x30  48       APTR tf_CharKern;
                   /* Starting character inside character
                      container */
0x34  52    }
```



6.6.9 GELs and sprites

AddAnimOb **Inserts AnimOb in GEL list**

Syntax:

```
AddAnimOb (AnimOb, AnimKey, RastPort)
  -156        A0      A1       A2
struct AnimOb   *AnimOb;
struct AnimOb   *AnimKey;
struct RastPort *RastPort;
```

Description: This function inserts all bobs of an animation object in the previously initialized GEL (Graphic ELement) list of the RastPort. The object can then be drawn using the `DrawGList()` function.

Parameters:

AnimOb: Pointer to the user-defined animation object.

AnimKey: Internal pointer to animation objects. It must be zero on the first call. When using multiple animation objects, `AnimKey` points to the animation object inserted last.

RastPort: Address of the RastPort containing the `GelsInfo` structure.

See Also: `Animate(),DrawGList()`

AddBob **Inserts bob in GEL list**

Syntax:

```
AddBob (Bob,RastPort)
 -96     A0,   A1
struct Bob      *Bob;
struct RastPort *RastPort;
```

Description: This function adds a bob (Blitter OBject) to the GEL list of a RastPort.

Parameters:

Bob: Address of the bob to be inserted.

RastPort: Address of the RastPort containing the `GelsInfo` structure needed for the creation of the GEL list.

See Also: `AddVSprite(),DrawGList()`

AddVSprite **Inserts vsprite in GEL list**

Syntax:

```
AddVSprite (VSprite, RastPort)
    -102      A0        A1
struct VSprite  *VSprite;
struct RastPort *RastPort;
```

Description: This function inserts an initialized vsprite (virtual sprite) in the GEL list of the given RastPort for later drawing.

You cannot display more than eight vsprites at a vertical position. Two of these vsprites must be identical colors. This means that if you wish to display sprites of different colors, you can only present a maximum of four vsprites per raster line.

Parameters:

Vsprite: Address of the vsprites to be inserted into the GEL list.
RastPort: Address of the RastPort in which the GEL list is defined.

See Also: `DrawGList()`

Animate — Animates animation objects

Syntax:

```
Animate (AnimKey, RastPort)
  -162      A0       A1
struct AnimOb   *AnimKey;
struct RastPort *RastPort;
```

Description: This function animates the animation object previously inserted using the `AnimOb()` function. Variables used by AnimObs must be defined before using animate. These variables specify data for the X and Y positions of the AnimOb, speed of movement, different versions of the AnimOb to simulate animation and more (see the Abacus book **Amiga Graphics Inside & Out** for additional information on AnimOb variables, as well as C language implementation of these objects).

Parameters:

AnimKey: Pointer to the `AnimKey` that may not be changed after the last `AddAnimOb()` function.
RastPort: Address of the RastPort which defines the GEL list.

See Also: `AddAnimOb()`

ChangeSprite — Changes sprite's appearance

Syntax:

```
ChangeSprite (ViewPort, Sprite, SpriteData)
    -420          A0       A1         A2
struct ViewPort     *ViewPort;
struct SimpleSprite *Sprite;
struct SpriteDaten  *SpriteData;
```

Description: This function changes the appearance of a hardware sprite. The `SimpleSprite` structure which contains the number of the corresponding hardware sprite specifies which sprite should be changed.

Unfortunately the `SpriteData` structure cannot be based in an include file so that the user must deal with it himself through programming. Look at the following structure:

```
struct SpriteData
     {
          UWORD posctl[2];
          UWORD Daten[Height][2];
          UWORD Reserved[2]; /* = 0,0 */
     }
```

The data array in this structure which contains the actual two-dimensional bit pattern of the sprite has other dimensions corresponding to the bit pattern. This must be determined by the user.

Parameters:

ViewPort:	Address of the ViewPort if the sprite should be positioned relative to the View or to the ViewPort, or a value of zero.
Sprite:	Address of the `SimpleSprite` structure which is initialized by the `GetSprite()` function.
SpriteData:	Address of the `SpriteData` structure that contains the sprite's bit pattern.

See Also: `GetSprite(),FreeSprite(),MoveSprite()`

DoCollision — Tests GELs on collisions

Syntax:

```
DoCollision (RastPort)
    -108        A1
struct RastPort *RastPort;
```

Description: This function tests for GEL/GEL collisions that automatically call the respective collision routine. The function should be called following each gel movement.

GELs (bobs and sprites) must be sorted according to X and Y coordinates in increasing order (`SortGList()`).

Parameter: RastPort: Address of the RastPort that contains the `GelsInfo` structure containing the GELs.

See Also: `SortGList(),SetCollision()`

DrawGList — Displays GELs

Syntax:

```
DrawGList (RastPort, ViewPort)
    -114      A1        A0
struct RastPort *RastPort;
struct ViewPort *ViewPort;
```

Description: This routine draws all of the bobs of the given RastPort's GEL list and creates the Copper lists for vsprite display. The vsprites do not appear on the screen immediately after `DrawGList();MakeVPort()`, `MrgCop()` and `LoadView()` must be called after `DrawGList()` to display the vsprites. `MakeScreen()` and `RethinkDisplay()` are called for Intuition screens.

Parameters:

RastPort: Pointer to the RastPort that contains the `GelsInfo` structure.

ViewPort: Address of the ViewPort in which the vsprites should appear, and for which the vsprite Copper lists should be calculated.

See Also: `MakeVPort(),MrgCop(),LoadView()`

FreeGBuffers — Frees GEL buffers

Syntax:

```
FreeGBuffers (AnimObject, RastPort, DoubleBuffer)
      -600        A0         A1            D0
struct AnimOb    *AnimObject;
struct RastPort *RastPort;
BOOL              DoubleBuffer;
```

Description: This function frees all of the buffers of an animation object previously allocated using `GetGBuffers()`.

Parameters:

AnimObject: Address of the animation object for which the GEL buffer was allocated using `GetGBuffers()`.

RastPort: Address of the RastPort which defines the `GelsInfo` structure.

DoubleBuffer: Indicates status of additional buffers used for storing bobs. TRUE indicates that a double buffer allocated by `GetGBuffers()` has been freed.

See Also: `GetGBuffers(),InitGBuffers()`

FreeSprite — Returns sprite to system

Syntax:

```
FreeSprite (SpriteNumber)
-414         D0
SHORT SpriteNumber;
```

Description: This function returns a sprite to the system from a program or task.

Parameter:

SpriteNumber: Number of the sprite to be freed, as declared in `GetSprite()`.

Warning: Do not attempt to free sprites that were not previously declared by you using `GetSprite()`.

See Also: `GetSprite(),ChangeSprite(),MoveSprite()`

GetGBuffers — Gets GEL buffers

Syntax:

```
Status = GetGBuffers (AnimObject, RastPort, DoubleBuffer)
   D0           -168        A0         A1,          D0
BOOL              Status;
struct AnimOb   *AnimObject;
struct RastPort *RastPort;
BOOL             DoubleBuffer;
```

Description: This function allocates all of the buffers needed for the bobs of an animation object. In addition to the `SaveBuffer`, the `BorderLine` and the `CollMask ( = ImageShadow)` functions, `DoubleBuffer == TRUE` allocates, double buffering used by the animation objects.

Parameters:

AnimObject: Address of the animation object for whose bobs the buffer should be allocated.

RastPort: Address of the RastPort which defines the `GelsInfo` structure containing the AnimObs and bobs.

DoubleBuffer: Indicates status of additional buffers used for storing bobs. TRUE indicates that a double buffer allocated by `GetGBuffers()` has been allocated in addition to other buffers.

Result: Status: Returns TRUE if enough memory could be allocated from `GetGBuffers()` to provide memory for all of the buffers.

Comments: You cannot guarantee that memory locations can be freed from all of the previously allocated buffers. GetGBuffers() can also be used to "refresh" memory.

See Also: `FreeGBuffers(),InitGBuffers()`

GetSprite — Allocates sprite

Syntax:

```
SpriteNumber = GetSprite (Sprite, DesSprite)
        D0        -408        A0        D0
SHORT                   SpriteNumber;
struct SimpleSprite *Sprite;
SHORT                   DesSprite;
```

Description: This function enables a hardware sprite, which can then be added to your own applications.

Parameters: Sprite: Address of the `SimpleSprite` structure to be allocated. This can be processed further using `ChangeSprite()` and `MoveSprite()`.

DesSprite: Number of the sprite that you want used (0-7), or -1 if the sprite number makes no difference to you.

Result: SpriteNumber: Returns the number of the assigned sprite or the value —1 if the desired sprite or no sprite at all could be allocated.

Comments: The Amiga reads the `GfxBase.SpriteReserved` variable to find which sprites are allocated. `GfxBase.SpriteReserved == 3` means that sprites 1 and 2 are already in use by other tasks. If you have received an assigned sprite, pay attention to the sprite number because `FreeSprite()` will need this number for freeing the sprite when the program ends.

See Also: `FreeSprite(),ChangeSprite()`

InitAnimate — Initializes AnimKey

Syntax:

```
InitAnimate (AnimKey)
(Macro)
struct AnimOb *AnimKey;
```

Description: This macro assigns a value to the `AnimKey` parameter, which is on the first call of `AddAnimOb()`. `InitAnimate()` can be found in the include file `graphics/gels.h`.

Parameter: AnimKey: Address of the `AnimKey` that must be given with each call of `AddAnimOb()`.

See Also: `AddAnimOb()`

InitGels — Initializes GEL list

Syntax:

```
InitGels (ListStart, ListEnd, GelsInfo)
 -120        A0        A1       A2
struct VSprite  *ListStart,
                *ListEnd;
struct GelsInfo *GelsInfo;
```

Description: This function initializes a `GelsInfo` structure, which manages vsprites and bobs in a user-defined `Vsprite` structure. All of the `Vsprite` structures, including those from bobs and vsprites alike, are organized into a linked list (GEL list).

The `Vsprite` structures are placed in a linked list to allow sorting through `SortGList()` by Y and X coordinates. After initialization of the `GelsInfo` structure this must be added to the RastPort (`RastPort.GelsInfo = GelsInfo`).

The `AddBob()` and `AddVSprite()` functions can be assigned the RastPort structure instead of the `GelsInfo` structure.

Parameters:

ListStart, ListEnd: Addresses of the two `Vsprite` structures that represent the start and the end of the GEL list. These `Vsprite` structures cannot be used for bobs or vsprites.

GelsInfo: the `GelsInfo` structure to be initialized.

See Also: `SortGList()`

InitGMasks — Initializes all GEL buffers

Syntax:

```
InitGMasks (AnimationObject)
    -174         A0
struct AnimOb *AnimationObject;
```

Description: This function initializes all of the buffers previously allocated for the bobs of an animation object. A double buffer parameter does not need to be given here as with `GetGBuffers()` and `FreeGBuffers()`, because `InitGMasks()` knows if the animation object or its bobs require double buffer operation.

Parameter:

AnimOb: Address of the animation object whose buffer (`ImageShadow`, `CollMask`, `Borderline`, etc.) should be initialized.

See Also: `InitMasks()`, `GetGBuffers()`, `FreeGBuffers()`

InitMasks — Initializes bob or vsprite buffers

Syntax:

```
InitMasks (Vsprite)
    -126     A0
struct VSprite *Vsprite;
```

Description: This function initializes the different buffers of a bob or vsprite. The border line of a bob/vsprite is configured so that all of the bit pattern lines of the bob/vsprite are joined by `OR` and saved in `BorderLine`.

When a bob is used in the `DoubleBuffer` operation, all of the other bobs must be supported by the `GelsInfo` structure as well as the `DoubleBuffer` operation.

The use of the bob in the `DoubleBuffer` operation is realized so that each bob has a `DBUfPacket` allocated. This later contains the saved background of the second bit-map, while `SaveBuffer`, like in the normal operation, contains the background of the first bit-map.

See Also: `InitGels()`

MoveSprite — Moves a sprite

Syntax:

```
MoveSprite (ViewPort, Sprite, x, y)
    -426       A0        A1   D0 D1
struct ViewPort     *ViewPort;
struct SimpleSprite *Sprite;
SHORT               x,y;
```

Description: This function places a sprite at the given position in the ViewPort, when specified. If the ViewPort is not specified, the sprite is positioned relative to the View. Unlike vsprites, the Copper lists do not have to be recalculated for normal hardware sprites. This is done by the `MoveSprite()` and `ChangeSprite()` functions. All sprites, including vsprites, can only be moved around pixels the size of a low-resolution screen. Sprite resolution is the same as low resolution. If you enable HIRES or LACE mode in your ViewPort, the size and resolution of the sprites remains unchanged. To move a sprite away you must give a minimum position change of two pixels.

Parameters:

ViewPort: Address of the ViewPort in which the sprite should be presented. Should the sprite be presented relative to the View, enter a value of zero for the ViewPort.

Sprite: Address of the `SimpleSprite` structure that should be written in closer to the moved sprite.

x,y: X and Y coordinates of the new sprite position.

See Also: `ChangeSprite(),GetSprite()`

RemBob — Removes bob from screen

Syntax:

```
RemBob (Bob)
(Macro)
struct Bob *Bob;
```

Description: This macro suppresses display of the specified bob. The next `DrawGList()` function call ignores the bob. This macro is defined in the include file `graphics/gels.h` and ensures that the `BOBSAWAY` flag is set in the bob.

Parameter:

Bob: Address of the bob that should be ignored on the next `DrawGList()` call.

Comments: Clearing the `BOBSAWAY` flag (`Bob.flags &=~ BOBSAWAY;`) again displays the bob.

See Also: `AddBob()`

RemIBob — Removes bobs from screen and GEL list

Syntax:

```
RemIBob (Bob, RastPort, ViewPort)
 -132     A0     A1        A2
struct Bob      *Bob;
struct RastPort +RastPort;
struct ViewPort *ViewPort;
```

Description: This function removes a bob from the GEL list and the screen. Removal from the screen is synchronized with the electron beam location given by the ViewPort.

Parameters:

Bob: Address of the bob to be removed.

RastPort: Address of the RastPort in which the `GelsInfo` structure is defined.

ViewPort: Address of the ViewPort in which the bob is presented.

See Also: `RemBob()`

RemVSprite — Removes vsprites

Syntax:

```
RemVSprite (VSprite)
    -138       A0
struct VSprite *VSprite;
```

Description: This function removes a vsprite from the GEL list and from the screen. This happens immediately, as with `RemIBob()`, but here you do not have to give a ViewPort for electron beam synchronization. After `RemVSprite()` the corresponding Copper list is recalculated and the electron beam synchronization compensates automatically.

Parameter:

VSprite: Address of the vsprite that should be removed from the GEL list. Because the GEL list consists of `Vsprite` structures, no RastPort containing the address of the `GelsInfo` structure is needed.

See Also: `AddVSprite()`

SetCollision — Determines collision routine

Syntax:

```
SetCollision (Number, Routine, GelsInfo)
      -144       D0       A0       A1
ULONG             Number;
VOOID           (*Routine())
struct GelsInfo *GelsInfo;
```

Description: This function inserts a routine that should be called when two GELs collide, in the corresponding memory `GelsInfo` structure (`GelsInfo.collHandler`). After two GELs collide, an `AND` combination joins the `HitMask` of one GEL and the `MeMask` of the other GEL. The resulting bit indicates the number of the routine (0-15) to which the program should jump. Routine 0 is always called when a GEL collides with the border, determined by the variables `GelsInfo.leftmost/rightmost/topmost/bottommost`.

The remaining routines (1-15) specify the addresses of the `Vsprite` structures in the collided GELs. Only routine 0 (GEL/border collision) contains a pointer to the vsprite of the GEL, and a flag that designates with which border the GEL has collided (LEFTHIT, RIGHTHIT, TOPHIT, BOTTOMHIT—see `graphics/collide.h`).

Parameters:

Number:	Number of the collision routine.
Routine:	Pointer to the collision routine.
GelsInfo:	Pointer to the `GelsInfo` structure of the GEL.

SortGList — Sorts GEL list

Syntax:

```
SortGList (RastPort)
    -150      A1
struct RastPort *RastPort;
```

Description: This function sorts the entries in the GEL list of the given RastPort. The GELs are sorted so that those with the smallest Y coordinates are at the beginning of the list and those with the largest Y coordinates are at the end of the list. If some GELS have matching Y coordinates, the X coordinate is the deciding factor, and the X coordinates are also sorted in increasing order. This sorting maintains optimum speed in accessing hardware sprites for vsprite generation. The advantage to bobs is that fewer bobs can be drawn before the electron beam, minimizing flickering. The drawing is done from `DrawGList()`.

Parameter: RastPort: Address of the RastPort containing the `GelsInfo` structure storing bob and vsprite data.

See Also: `DrawGList()`

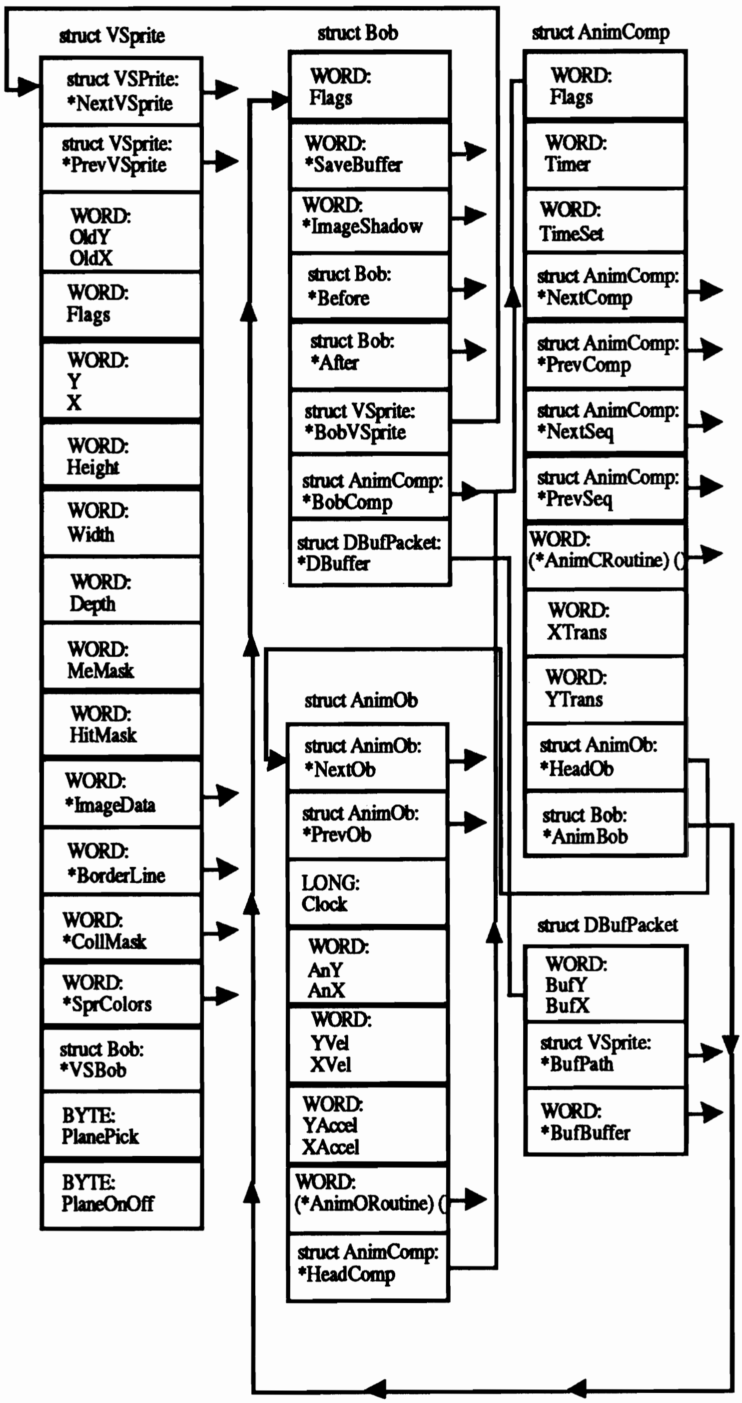
Structures:

```
Offset    Structures
------    ----------

          struct VSprite <graphics/gels.h>
          {
0x00    0       struct VSprite *NextVSprite;
0x04    4       struct VSprite *PrevVsprite;
                    /* For linking to the GEL List */
0x08    8       struct VSprite *DrawPath;
                   /* Presentation path */
```

```
0x1c   12        struct VSprite *ClearPath;
                     /* Clearing path */
0x10   16        WORD OldY,
0x12   18             OldX;
                     /* Old position */
0x14   20        WORD Flags;
0x16   22        WORD Y,
0x18   24             X;
                     /* Current position */
0x1a   26        WORD Height;
                     /* Height */
0x1c   28        WORD Width;
                     /* Width */
0x1e   30        WORD Depth;
                     /* Depth (VSprite = 1) */
0x20   32        WORD MeMask;
0x22   34        WORD HitMask;
                     /* Collision bits */
0x26   38        WORD *ImageData;
                     /* Address of bit pattern */
0x2a   42        WORD *BorderLine;
0x2e   46        WORD *CollMask;
                     /* Collision mask */
0x32   50        WORD *SVSprite */
0x36   54        struct Bob *VSBob;
                     /* Bob address */
0x37   56        BYTE PlanePick;
0x38   57        BYTE PlaneOnOff;
                     /* Which bit-planes are related */
0x3a   58        VUserStuff VUserExt;
                     /* User-defined */
               }

               struct Bob <graphics/gels.h>
               {
0x00    0        WORD Flags;
0x02    2        WORD *SaveBuffer;
                     /* Address of background memory */
0x06    6        WORD *ImageShadow;
                     /* OR all of the planes */
0x0a   10        struct Bob *Before;
0x0e   14        struct Bob *After;
                     /* Previously set character order */
0x12   18        struct VSprite *BobVSprite;
                     /* Address of the BobVSprite */
0x16   22        struct AnimComp *BobComp;
                     /* Animation component addresses of Bob
                        sequence in an AnimComp*/
0x1a   26        struct DBuffPacket *DBuffer;
                     /* For use in DoubleBuffer BitMaps */
0x1e   30        BUserStuff BUserExt;
                     /* User-defined */
               }

               struct DBufPacket <graphics/gels.h>
               {
0x00   0         WORD BufY,
0x02   2              BufX;
                     /* Background position in second bit-map */
0x04   4         struct VSprite *BufPath;
0x08   8         WORD *BufBuffer;
                     /* Address of background memory */
```

```
0x0c  12  }

          struct AnimComp <graphics/gels.h>
          {
0x00   0        WORD Flags;
0x02   2        WORD Timer;
0x04   4        WORD TimeSet;
0x06   6        struct AnimComp *NextComp;
0x0a  10        struct AnimComp *PrevComp;
                    /* For component linking */
0x0e  14        struct AnimComp *NextSeq;
0x12  18        struct AnimComp *PrevSeq;
                    /* For linking sequences */
0x14  22        WORD (*AnimCRoutine)();
                    /* Routine to be called */
0x1a  26        WORD XTrans;
0x1c  28        WORD YTrans;
                    /* Speed */
0x1e  30        struct AnimOb *HeadOb;
                    /* Interface to AnimObject */
0x22  34        struct Bob *AnimBob;
                    /* Bob address */
0x26  38  }

          struct AnimOb <graphics/gels.h>
          {
0x00   0        struct AnimOb *NextOb,
0x04   4                      *PrevOb;
                    /* For linking */
0x08   8        LONG Clock;
0x0c  12        WORD AnOldY,
0x0e  14             AnOldX;
                    /* Old position */
0x10  16        WORD AnY,
0x12  18             AnX;
                    /* Current position */
0x14  20        WORD YVel,
0x16  22             XVel;
                    /* Speed */
0x18  24        WORD YAccel,
0x1a  26             XAccel;
                    /* Acceleration */
0x1c  28        WORD RingYTrans,
0x1e  30             RingXTrans;
                    /* Ringtrigger speed */
0x20  32        WORD (*AnimORoutine)();
                    /* Routine to be called */
0x24  36        struct AnimComp *HeadComp;
                    /* Address of first component */
0x28  40        AUserStuff AUserExt;
                    /* User-defined */
          }

          struct SimpleSprite <graphics/sprite.h>
          {
0x00   0        UWORD *posctldata;
                    /* Address of the sprite data */
0x04   4        UWORD height;
                    /* Height */
0x06   6        UWORD x,
0x08   8              y;
                    /* Position */
```

```
0x0a  10        UWORD num;
                    /* Number of the sprite */
0x0c  12   }

           struct GeslsInfo <graphics/rastport.h>
           {
0x00   0        UBYTE SprRsrvd;
                    /* Reserved sprites */
0x01   1        UBYTE Flags;
0x02   2        struct VSprite *gelHead,
0x06   6                       *gelTail;
                    /* Beginning and end of GEL List */
0x0a  10        WORD *nextLine;
0x0e  14        WORD **lastColor;
                    /* VSprite colors presented last */
0x12  18        struct collTable *collHandler;
                    /* Collision Handler */
0x16  22        short leftmost,
0x18  24              rightmost,
0x1a  26              topmost,
0x1c  28              bottommost;
                    /* Rectangle for border collision */
0x1c  30        APTR firstBlissObj,
0x22  34             lastBlissObj;
0x26  38   }

           struct CollTable <graphics/gels.h>
           {
0x00   0        int (*collPtrs[16])();
                    /* 16 collision routine addresses */
0x40  64 }
```


struct VSprite
struct VSPrite: *NextVSprite
struct VSprite: *PrevVSprite
WORD: OldY OldX
WORD: Flags
WORD: Y X
WORD: Height
WORD: Width
WORD: Depth
WORD: MeMask
WORD: HitMask
WORD: *ImageData
WORD: *BorderLine
WORD: *CollMask
WORD: *SprColors
struct Bob: *VSBob
BYTE: PlanePick
BYTE: PlaneOnOff
struct Bob
WORD: Flags
WORD: *SaveBuffer
WORD: *ImageShadow
struct Bob: *Before
struct Bob: *After
struct VSprite: *BobVSprite
struct AnimComp: *BobComp
struct DBufPacket: *DBuffer
struct AnimOb
struct AnimOb: *NextOb
struct AnimOb: *PrevOb
LONG: Clock
WORD: AnY AnX
WORD: YVel XVel
WORD: YAccel XAccel
WORD: (*AnimORoutine) ()
struct AnimComp: *HeadComp
struct AnimComp
WORD: Flags
WORD: Timer
WORD: TimeSet
struct AnimComp: *NextComp
struct AnimComp: *PrevComp
struct AnimComp: *NextSeq
struct AnimComp: *PrevSeq
WORD: (*AnimCRoutine) ()
WORD: XTrans
WORD: YTrans
struct AnimOb: *HeadOb
struct Bob: *AnimBob
struct DBufPacket
WORD: BufY BufX
struct VSprite: *BufPath
WORD: *BufBuffer

6.7 The DiskFont library

The DiskFont library makes it possible to quickly and easily load additional fonts from disk. There are two functions for this, to which Kickstart1.3 adds two more functions.

DiskFont library functions

AvailFonts **Creates list of available fonts**

Syntax:

```
Error = AvailFonts(Buffer,Length,Type)
   D0      -36      A0     D0    D1
LONG Error;
UBYTE *Buffer;
LONG Length;
LONG Type;
```

Description: This function fills a data buffer with information about the fonts available on disk or in memory. Disk fonts must be loaded using the `OpenDiskFont` function, while fonts in memory can be opened with the `OpenFont` function (graphics library).

Parameters:

Buffer: Pointer to the data buffer. This fills with the `AvailFontsHeader` structure.

Length: Length of the data buffer in bytes.

Type: Indicates the location of the fonts about which you want information. The `Type` parameter can contain `AFF_MEMORY`, `AFF_DISK`, or both. This system then returns the fonts available in memory, on disk or in both media.

Result:

Error: Returns 0 if the function executes without an error, or the number of bytes by which the data buffer fell short in trying to load the information. This invalidates the contents of the data buffer.

Comments: If you give (AFF_MEMORY | AFF_DISK) for the Type parameter, the system examines fonts on disk as well as in memory. Some fonts may be duplicated between memory and disk; the Type array indicates this (see the TextAttr structure).

The AvailFonts function checks the FontContents file for the font name, but it doesn't check the disk for the font file itself. If the font is missing, opening the non-existent font using the OpenDiskFont function returns an error message.

See Also: FontsContents structure

DisposeFontContents — Frees FontsContentsHeader

Syntax:

```
DisposeFontContents (FontContentsHeader)
          -48                 A1
```

Description: This function frees the memory that was allocated by the NewFontContents function.

Parameter: FontContentsHeader:
Pointer to the FontContentsHeader structure returned by the NewFontsContents function.

Warning: If you state the address of a FontContentsHeader that is not employed by the NewFontContents function, the system may crash.

See Also: NewFontContents

NewFontContents — Reads font data

Syntax:

```
FontContentsHeader = NewFontContents(FontsLock,FontName)
        D0                  -42            A0        A1
```

Description: This function adds a FontsContentsHeader structure in memory. All data found in the directory passed by FontsLock, and whose name reads FontName is saved in this structure.

Parameters:

FontsLock: Directory lock on the FontContents file and the subdirectory containing the current font (usually the FONTS directory).

FontName: Font filename plus a file extension of .font, the name of the FontContents file.

Result: FontContentsHeader:
Pointer to the FontContentsHeader structure.

Exceptions: If an error occurs (file not found, etc.), a value of zero is returned.

See Also: `DisposeFontContents`

OpenDiskFont — Loads font from disk

Syntax:

```
CharSet = OpenDiskFont(textAttr)
   D0             -30      A0
struct Font *CharSet;
struct TextAttr *textAttr;
```

Description: This function loads the font described by the `TextAttr` structure from disk and returns a pointer to this font. When the font is no longer needed, call the `CloseFont` function to save memory. If the font already exists in memory, the system returns a pointer without loading the font a second time.

Parameter: TextAttr: `TextAttr` structure describing the font to be loaded.

Result: CharSet: Pointer to the font.

Exceptions: If the font is not on disk or in memory, a value of zero is returned.

See Also: `CloseFont`, `SetFont`

Structures:

```
struct FontContentsHeader <libraries/diskfont.h>
{
0x00 0  UWORD fch_FileID;      /* FCH_ID */
0x02 2  UWORD fch_NumEntries;  /* Number of entries */
0x04 4
   /*   struct FontContents fch_FC[]; */
};

struct FontContents <libraries/diskfont.h>
{
0x000   0  char  fc_FileName[MAXFONTPATH];
0x100 256  UWORD fc_YSize;  /* Font height */
0x102 258  UBYTE fc_Style;  /* Text type */
0x103 259  UBYTE fc_Flags;  /* Font type */
0x104 260
};
MAXFONTPATH 256   /* Inclusive null byte */
FCH_ID      0x0f00

struct AvailFontsHeader <libraries/diskfont.h>
{
0x00 0  UWORD afh_NumEntries;/* Number of entries */
0x02 2
   /*   struct   AvailFonts afh_AF[]; */
};

struct AvailFonts <libraries/diskfont.h>
{
0x00  0  UWORD af_Type;    /* MEMORY or DISK */
0x02  2  struct   TextAttr af_Attr;
0x0A 10
};
```

Type (af_Type):

```
AFF_MEMORY  1
AFF_DISK    2
struct TextAttr <graphics/text.h>
{
0x00 0  STRPTR   ta_Name;
0x04 4  UWORD    ta_YSize;  /* Font height */
0x06 6  UBYTE    ta_Style;  /* Text style */
0x07 7  UBYTE    ta_Flags;  /* Font type */
0x08 8
};
```

Text styles (ta_Style):

```
FS_NORMAL       0
FSF_EXTENDED    (1<<3)
FSF_ITALIC      (1<<2)
FSF_BOLD        (1<<1)
FSF_UNDERLINED  (1<<0)
```

Character set-Typen (ta_Flags):

```
FPF_ROMFONT  (1<<0) /* Font found in the ROM */
FPF_DISKFONT (1<<1) /* Font found on disk */
FPF_REVPATH  (1<<2)
FPF_TALLDOT  (1<<3) /* 640x200 resolution */
FPF_WIDEDOT  (1<<4) /* 320x400 resolution */
FPF_PROPORTIONAL (1<<5)  /* Proportional font */
FPF_DESIGNED (1<<6)
FPF_REMOVED  (1<<7) /* Font removed */
struct TextFont <graphics/text.h>
{
0x00  0  struct Message tf_Message;
0x14 20  UWORD    tf_YSize;     /* Character height */
0x16 22  UBYTE    tf_Style;
0x17 23  UBYTE    tf_Flags;
0x18 24  UWORD    tf_XSize;     /* Character width */
0x1A 26  UWORD    tf_Baseline;
0x1C 28  UWORD    tf_BoldSmear;
0x1E 30  UWORD    tf_Accessors;
0x20 32  UBYTE    tf_LoChar;    /* First character */
0x21 33  UBYTE    tf_HiChar;    /* Last character */
0x22 34  APTR     tf_CharData;
0x26 38  UWORD    tf_Modulo;
0x28 40  APTR     tf_CharLoc;
0x2C 44  APTR     tf_CharSpace;
0x30 48  APTR     tf_CharKern;
0x34 52
};
```

6.8 The math libraries

The operating system of the Amiga gives you four different math libraries: Two for single precision (FFP) numbers, and two for double precision (IEEE) numbers. One contains the basic calculations needed, and the other contains trigonometric functions.

6.8.1 The math library

This library offers two different floating-point numeric formats: FFP and IEEE. The functions for FFP floating-point numbers are in the math library and are the computing basics. When you program in C you usually don't need to worry about special floating-point calculations, because most C compilers include these functions in an onboard math library.

Changing formats poses some problems. For example, the Aztec C compiler allows global formatting for one module, while the earlier versions of the Lattice C compilers didn't include this option (this was corrected with Version 4.0 of Lattice C).

What do you do when you want to use both single precision and double precision numbers in your program? You must use the libraries and separate the elements of the equation. Amiga assembly language accepts floating-point math much more easily than other computers, where the programmer must program the functions on his/her own.

About the layout of these functions. Each function states a syntax, a description of the function, the assembler condition code and any additional data as needed.

Math library functions

SPAbs — Absolute value

Syntax:

```
result = SPAbs(value)
  D0      -54    D0
FLOAT result;
FLOAT value;
```

Description: Calculates an absolute value from `value`.

Assembler condition code:

```
N = 0
Z = 1, if result = 0
V = 0
C = not defined
X = not defined
```

SPAdd — Addition

Syntax:

```
result = SPAdd(value1,value2)
D0         -66     D1     D0
FLOAT result;
FLOAT value1,value2;
```

Description: Adds `value1` and `value2`.

Assembler condition code:

```
N = 1 if result < 0
Z = 1 if result = 0
V = 1 if overflow
C = not defined
X = not defined
```

SPCmp — Compares two numbers

Syntax:

```
result = SPCmp(value1,value2)
   D0      -42      D1     D0
LONG result;
FLOAT value1,value2;
```

Description: Compares `value1` with `value2`. The result is:

```
+1 if value1 < value2
0 if value1 = value2
-1 if value1 > value2
```

Assembler condition code:

```
GT, if value2 > value1
GE, if value2 >= value1
EQ, if value2 = value1
NE, if value2 <> value1
LT, if value2 < value1
LE, if value2 <= value1
```

SPDiv — Division

Syntax:

```
result = SPDiv(value1,value2)
  D0      -84      D1      D0
FLOAT result;
FLOAT value1,value2;
```

Description: Divides `value2` by `value1`.

Assembler condition code:

```
N = 1 if result < 0
Z = 1 if result = 0
V = 1 if overflow
C = not defined
X = not defined
```

SPFix — Converts FFP to integer format

Syntax:

```
result = SPFix(value)
  D0      -30     D0
LONG result;
FLOAT value;
```

Description: Converts an FFP number to an integer (two's complement).

Assembler condition code:

```
N = 1 if result < 0
Z = if result = 0
V = if overflow
C = not defined
X = not defined
```

SPFlt — Converts integer to FFP format

Syntax:

```
result = SPFlt(value)
   D0     -36   D0
FLOAT result;
LONG value;
```

Description: Converts an integer (two's complement) to FFP format.

Assembler condition code:

```
N = 1 if result < 0
Z = 1 if result = 0
V = 0
C = not defined
X = not defined
```

SPMul — Multiplication

Syntax:

```
result = SPMul(value1,value2)
   D0     -78    D1      D0
FLOAT result;
FLOAT value1,value2;
```

Description: Multiplies `value1` by `value2`.

Assembler condition code:

```
N = 1 if result < 0
Z = if result = 0
V = if overflow
C = not defined
X = not defined
```

SPNeg — Swaps number sign

Syntax:

```
result = SPNeg(value)
   D0     -60    D0
FLOAT result;
FLOAT value;
```

Description: Swaps the sign from `value:`→ `result = -value`.

Assembler condition code:

```
N = 1 if result < 0
Z = if result = 0
V = 0
C = not defined
X = not defined
```

SPSub — Subtraction

Syntax:

```
result = SPSub(value1,value2)
   D0    -72     D1     D0
FLOAT result;
FLOAT value1,value2;
```

Description: Subtracts `value1` from `value2`:→ `result = value2 - value1`.

Assembler condition code:

```
N = 1 if result < 0
Z = if result = 0
V = if overflow
C = not defined
X = not defined
```

SPTst — Compares number with zero

Syntax:

```
result = SPTst(value)
   D0     -48   D1
LONG result;
FLOAT value;
```

Description: Tests if `value` is zero. The result is:

```
+1 if value > 0
0 if value = 0
-1 if value < 0
```

Assembler condition code:

```
N = 1 if result < 0
Z = if result = 0
V = 0
C = not defined
X = not defined
```

FFP — Format

Syntax:

```
MMMMMMMM MMMMMMMM MMMMMMMM SEEEEEEE
31 23 15 7
```

Meaning:

```
M = 24-bit mantissa
S = sign
E = 7-bit exponent
```

Value range (decimal):

```
 9.22337177 * 10^18 > +value > 5.42101070 * 10^-20
-9.22337177 * 10^18 < -value < -2.71050535 * 10^-20
```

Value range (binary):

```
 0.FFFFFF * 2^3F > +value > 0.800000 * 2^-3F
-0.FFFFFF * 2^3F < -value < -0.800000 * 2^-40
```

6.8.2 The MathTrans library

The MathTrans library gives you the transcendental functions needed for single precision. Everything concerning format in the math library applies here. Single precision floating-point numbers are usually in FFP format.

MathTrans library functions

SPAcos — Calculates arccosine

Syntax:

```
result = SPAcos(value)
   D0    -120    D0
FLOAT result;
FLOAT value;
```

Description: Calculates the arccosine of `value`.

Assembler condition code:

```
N = 0
Z = if result = 0
V = 0
```

```
C = not defined
X = not defined
```

SPAsin — Calculates arcsine

Syntax:

```
result = SPAsin(value)
   D0      -114   D0
FLOAT result;
FLOAT value;
```

Description: Calculates the arcsine of `value`.

Assembler condition code:

```
N = 0
Z = if result = 0
V = 0
C = not defined
X = not defined
```

SPAtan — Calculates arctangent

Syntax:

```
result = SPAtan(value)
   D0      -30    D0
FLOAT result;
FLOAT value;
```

Description: Calculates the arctangent of `value`.

Assembler condition code:

```
N = 0
Z = if result = 0
V = 0
C = not defined
X = not defined
```

SPCos — Calculates cosine

Syntax:

```
result = SPCos(value)
  D0      -42    D0
FLOAT result;
FLOAT value;
```

Description: Calculates the cosine of `value`.

Assembler condition code:

```
N = 1 if result < 0
Z = 1 if result = 0
V = 1 if value is too large
C = not defined
X = not defined
```

SPCosh — Calculates hyperbolic cosine

Syntax:

```
result = SPCosh(value)
   D0     -66    D0
FLOAT result;
FLOAT value;
```

Description: Calculates the hyperbolic cosine of `value`.

Assembler condition code:

```
N = 1 if result < 0
Z = 1 if result = 0
V = 1 if overflow
C = not defined
X = not defined
```

SPExp — Calculates e to the x power

Syntax:

```
result = SPExp(value)
   D0     -78    D0
FLOAT result;
FLOAT value;
```

Description: Calculates e raised to the power of `value`.

Assembler condition code:

```
N = 0
Z = 1 if result = 0
V = 1 if overflow
C = not defined
X = not defined
```

SpFieee — Converts IEEE format to FFP

Syntax:

```
result = SPFieee(value)
   D0    -108     D0
FLOAT result;
FLOAT value; /* IEEE standard format */
```

Description: Converts simple precision IEEE standard format into FFP format.

Assembler condition code:

```
N = not defined
Z = 1 if result = 0
V = 1 if overflow
C = not defined
X = not defined
```

SPLog — Calculates natural logarithm

Syntax:

```
result = SPLog(value)
   D0     -84    D0
FLOAT result;
FLOAT value;
```

Description: Calculates the natural logarithm of `value`.

Assembler condition code:

```
N = 1 if result < 0
Z = 1 if result = 0
V = 1 if value <= 0
C = not defined
X = not defined
```

SPLog10 — Calculates base 10 logarithm

Syntax:

```
result = SPLog10(value)
   D0     -126    D0
FLOAT result;
FLOAT value;
```

Description: Calculates the base 10 logarithm of `value`.

Assembler condition code:

```
N = 1 if result < 0
Z = 1 if result = 0
V = 1 if value <= 0
C = not defined
X = not defined
```

SPPow — Calculates x to the y power

Syntax:

```
result = SPPow(value1,value2)
   D0      -90    D0      D1
FLOAT result;
FLOAT value1,value2;
```

Description: Calculates `value1^value2`.

Assembler condition code:

```
N = 0
Z = 1 if result = 0
V = 1 if overflow or value1 < 0
C = not defined
X = not defined
```

SPSin — Calculates sine

Syntax:

```
result = SPSin(value)
   D0    -36    D0
FLOAT result;
FLOAT value;
```

Description: Calculates the sine of `value`.

Assembler condition code:

```
N = 1 if result < 0
Z = 1 if result = 0
V = 1 if value is too large
C = not defined
X = not defined
```

SPSincos — Calculates sine and cosine

Syntax:

```
result = SPSincos(value,adr_c)
  D0       -54       D0    D1
FLOAT result;
FLOAT value,*adr_c;
```

Description: Calculates the sine and cosine of `value`. Returns the sine immediately and saves the cosine in the variable to which `adr_c` points.

Assembler condition code:

```
N = 1 if result < 0
Z = 1 if result = 0
V = 1 if value is too large
C = not defined
X = not defined
```

SPSinh — Calculates hyperbolic sine

Syntax:

```
result = SPSinh(value)
   D0      -60    D0
FLOAT result;
FLOAT value;
```

Description: Calculates the hyperbolic sine of `value`.

Assembler condition code:

```
N = 1 if result < 0
Z = 1 if result = 0
V = 1 if overflow
C = not defined
X = not defined
```

SPSqrt — Calculates square root

Syntax:

```
result = SPSqrt(value)
   D0       -96    D0
FLOAT result;
FLOAT value;
```

Description: Calculates the root of `value`.

Assembler condition code:

```
N = 0
Z = 1 if result = 0
V = 1 if value < 0
C = not defined
X = not defined
```

SPTan — Tangent

Syntax:

```
result = SPTan(value)
   D0      -48    D0
FLOAT result;
FLOAT value;
```

Description: Calculates the tangent of `value`.

Assembler condition code:

```
N = 1 if result < 0
Z = 1 if result = 0
V = 1 if cos(value) = 0
C = not defined
X = not defined
```

SPTanh — Calculates hyperbolic tangent

Syntax:

```
result = SPTanh(value)
   D0       -72    D0
FLOAT result;
FLOAT value;
```

Description: Calculates the hyperbolic tangent of `value`.

Assembler condition code:

```
N = 1 if result < 0
Z = 1 if result = 0
V = 1 if overflow
C = not defined
X = not defined
```

SPTieee — Converts FFP into IEEE format

Syntax:

```
result = SPTieee(value)
   D0     -102    D0
FLOAT result;  /* Standard IEEE format */
FLOAT value;
```

Description: Converts FFP format into simple precision standard IEEE format.

Assembler condition code:

```
N = 1 if result < 0
Z = 1 if result = 0
V = not defined
C = not defined
X = not defined
```

FFP — Format

Syntax:

```
MMMMMMMM MMMMMMMM MMMMMMMM SEEEEEEE
31 23 15 7
```

Meaning:

```
M = 24 bit mantissa
S = sign
E = 7 bit exponent
```

Value range (decimal):

```
 9.22337177 * 10^18 > +value > 5.42101070 * 10^-20
-9.22337177 * 10^18 < -value < -2.71050535 * 10^-20
```

Value range (binary):

```
 0.FFFFFF * 2^3F > +value > 0.800000 * 2^-3F
-0.FFFFFF * 2^3F < -value < -0.800000 * 2^-40
```

6.8.3 The MathIeeeDoubBas library

The MathIeeeDoubBas library gives you double precision math functions. Double precision floating-point numbers are usually expected in IEEE format.

Operating system Version 1.3 runs the functions of this library seven times faster than before. In addition, this library then automatically supports a 68881 coprocessor once such a processor is installed in the system. A 68881 in conjunction with a 68020 is automatically recognized (under Kickstart 1.2 as well). When you find the 68881 alone in the computer, it must be accessed through the `MathIEEE.resource` of the operating system.

MathIeeeDoubBas library functions

IEEEDPAbs — Absolute value

Syntax:

```
result = IEEEDPAbs(value)
D0/D1     -54      D0/D1
DOUBLE result;
DOUBLE value;
```

Description: Calculates the absolute value of `value`.

Assembler condition code:

```
N = 0
Z = 1 if result = 0
V = 0
C = not defined
X = not defined
```

IEEEDPAdd — Addition

Syntax:

```
result = IEEEDPAdd(value1,value2)
D0/D1     -66      D0/D1   D2/D3
```

Description: Adds `value1` and `value2`.

Assembler condition code:

```
N = 1 if result < 0
Z = 1 if result = 0
V = 1 if overflow
C = not defined
X = not defined
```

IEEEDPCeil — Smallest integer equivalent

Syntax:

```
result = IEEEDPCeil(value)
D0/D1        -96    D0/D1
DOUBLE result;
DOUBLE value;
```

Description: Finds the smallest integer greater than or equal to `value`.

Assembler condition code:

Unknown.

IEEEDPCmp — Comparison

Syntax:

```
result = IEEEDPCmp(value1,value2)
  D0       -42       D0/D1  D2/D3
LONG result;
DOUBLE value1,value2;
```

Description: Compares `value1` with `value2`. The result is:

```
+1 if value1 > value2
0 if value1 = value2
-1 if value1 < value2
```

Assembler condition code:

```
GT if value > value2
GE if value1 >= value2
EQ if value1 = value2
NE if value1 <> value2
LT if value1 < value2
LE if value1 <= value2
```

IEEEDPDiv — Division

Syntax:

```
result = IEEEDPDiv(value1,value2)
D0/D1      -84      D0/D1  D2/D3
DOUBLE result;
DOUBLE value1,value2;
```

Description: Divides `value1` by `value2`.

Assembler condition code:

```
N = 1 if result < 0
Z = 1 if result = 0
V = 1 if overflow
C = not defined
X = not defined
```

IEEEDPFix — Converts IEEE to integer format

Syntax:

```
result = IEEEDPFix(value)
   D0      -30      D0/D1
LONG result;
DOUBLE value;
```

Description: Converts an IEEE number into integer format.

Assembler condition code:

```
N = 1 if result < 0
Z = 1 if result = 0
V = 1 if overflow
C = not defined
X = not defined
```

IEEEDPFloor — Largest whole number

Syntax:

```
result = IEEEDPFloor(value)
D0/D1          -90    D0/D1
DOUBLE result;
DOUBLE value;
```

Description: Finds the largest whole number less than or equal to `value`.

Assembler condition code:

Unknown.

EEEDPFlt — Converts integer to IEEE format

Syntax:

```
result = IEEEDPFlt(value)
D0/D1        -36     D0
DOUBLE result;
LONG value;
```

Description: Converts an integer into IEEE format.

Assembler condition code:

```
N = 1 if result < 0
Z = 1 if result = 0
V = 0
C = not defined
X = not defined
```

IEEEDPMul — Multiplication

Syntax:

```
result = IEEEDPMul(value1,value2)
D0/D1        -78     D0/D1  D2/D3
DOUBLE result;
DOUBLE value1,value2;
```

Description: Multiplies `value1` and `value2`.

Assembler condition code:

```
N = 1 if result < 0
Z = 1 if result = 0
V = 1 if overflow
C = not defined
X = not defined
```

IEEEDPNeg — Swaps number sign

Syntax:

```
result = IEEEDPNeg(value)
D0/D1        -60    D0/D1
DOUBLE result;
DOUBLE value;
```

Description: Sign change:→ `result = -value`.

Assembler condition code:

```
N = 1 if result < 0
Z = 1 if result = 0
V = 0
C = not defined
X = not defined
```

IEEEDPSub — Subtraction

Syntax:

```
result = IEEEDPSub(value1,value2)
D0/D1        -72     D0/D1  D2/D3
DOUBLE result;
DOUBLE value1,value2;
```

Description: Subtracts `value2` from `value1`:→ `result = value1 - value2`.

Assembler condition code:

```
N = 1 if result < 0
Z = 1 if result = 0
V = 1 if overflow
C = not defined
X = not defined
```

IEEEDPTst — Tests for zero

Syntax:

```
result = IEEEDPTst(value)
   D0         -48   D0/D1
LONG result;
DOUBLE value;
```

Description: Tests if `value` is zero. The result is:

```
+1 if value > 0
0 if value = 0
-1 if value < 0
```

Assembler condition code:

```
N = 1 if result < 0
Z = 1 if result = 0
V = 0
C = not defined
X = not defined
```

Structures:

```
MathIEEE.resource:
struct MathIEEE =
{
0x00  0  struct Node MathIEEE_node;
0x0E 14  UWORD  MathIEEE_Flags
0x10 16  ULONG  MathIEEE_BaseAddr    ;für 68881-Coprozessor
0x14 20  ULONG  MathIEEE_DblBasInit  ;für andere
0x18 24  ULONG  MathIEEE_DblTransInit
0x1C 28  ULONG  MathIEEE_SnglBasInit
0x20 32  ULONG  MathIEEE_SnglTransInit
0x24 36
}
```

6.8.4 The MathIeeeDoubTrans library

The MathIeeeDoubTrans library gives you the transcendental functions needed for double precision math (IEEEDP = IEEE Double Precision). Overall the double precision floating-point numbers are expected in IEEE format.

This library supports a 68881 processor once installed in the system, just like the MathIeeeDoubBas library. If just the 68881 is found in the computer, this must be accessed through the `MathIEEE.resource` in the operating system.

MathTrans library

IEEEDPAcos — Arccosine

Syntax:

```
result = IEEEDPAcos(value)
D0/D1       -120    D0/D1
DOUBLE result;
DOUBLE value;
```

Description: Calculates the arccosine of `value`.

IEEEDPAsin — Arcsine

Syntax:

```
result = IEEEDPAsin(value)
D0/D1        -114    D0/D1
DOUBLE result;
DOUBLE value;
```

Description: Calculates the arcsine of `value`.

IEEEDPAtan — Arctangent

Syntax:

```
result = IEEEDPAtan(value)
D0/D1         -30    D0/D1
DOUBLE result;
DOUBLE value;
```

Description: Calculates the arctangent of `value`.

IEEEDPCos — Cosine

Syntax:

```
result = IEEEDPCos(value)
D0/D1         -42   D0/D1
DOUBLE result;
DOUBLE value;
```

Description: Calculates the cosine of `value`.

IEEEDPCosh — Hyperbolic cosine

Syntax:

```
result = IEEEDPCosh(value)
D0/D1         -66    D0/D1
DOUBLE result;
DOUBLE value;
```

Description: Calculates the hyperbolic cosine of `value`.

IEEEDPExp — e raised to the x power

Syntax:

```
result = IEEEDPExp(value)
D0/D1        -78    D0/D1
DOUBLE result;
DOUBLE value;
```

Description: Calculates e raised to the power of `value`.

IEEEDPFieee — Converts IEEESP format to IEEEDP

Syntax:

```
result = IEEEDPFieee(value)
D0/D1       -108      D0
DOUBLE result;
FLOAT value; /* Single precision IEEE format */
```

Description: Converts a single precision IEEE number to double precision IEEE format.

IEEEDPLog — Natural logarithm

Syntax:

```
result = IEEEDPLog(value)
D0/D1       -84    D0/D1
DOUBLE result;
DOUBLE value;
```

Description: Calculates the natural logarithm of `value`.

IEEEDPLog10 — Base 10 logarithm

Syntax:

```
result = IEEEDPLog10(value)
D0/D1       -126       D0/D1
DOUBLE result;
DOUBLE value;
```

Description: Calculates the base 10 logarithm of `value`.

IEEEDPPow — x to the y power

Syntax:

```
result = IEEEDPPow(value2,value1)
D0/D1        -90    D2/D3  D0/D1
DOUBLE result;
DOUBLE value1,value2;
```

Description: Calculates `value1^value2`.

IEEEDPSin — Sine

Syntax:

```
result = IEEEDPSin(value)
D0/D1        -36    D0/D1
DOUBLE result;
DOUBLE value;
```

Description: Calculates the sine of `value`.

IEEEDPSincos — Sine and cosine

Syntax:

```
result = IEEEDPSincos(adr_c,value)
D0/D1          -54        A0  D0/D1
DOUBLE result;
DOUBLE value,*adr_c;
```

Description: Calculates the sine and cosine of `value`. The sine returns immediately and the cosine is saved in the variable to which `adr_c` points.

IEEEDPSinh — Hyperbolic sine

Syntax:

```
result = IEEEDPSinh(value)
D0/D1          -60   D0/D1
DOUBLE result;
DOUBLE value;
```

Description: Calculates the hyperbolic sine of `value`.

IEEEDPSqrt — Square root

Syntax:

```
result = IEEEDPSqrt(value)
D0/D1          -96   D0/D1
DOUBLE result;
DOUBLE value;
```

Description: Calculates the square root of `value`.

IEEEDPTan — Tangent

Syntax:

```
result = IEEEDPTan(value)
D0/D1         -48   D0/D1
DOUBLE result;
DOUBLE value;
```

Description: Calculates the tangent of `value`.

IEEEDPTanh — Hyperbolic tangent

Syntax:

```
result = IEEEDPTanh(value)
D0/D1        -72     D0/D1
DOUBLE result;
DOUBLE value;
```

Description: Calculates the hyperbolic tangent of `value`.

IEEEDPTieee — Converts IEEEDP into IEEESP format

Syntax:

```
result = IEEEDPTieee(value)
   D0       -102     D0/D1
FLOAT  result;  /* Single precision IEEE format */
DOUBLE value;
```

Description:

Converts a double precision IEEE number into single precision IEEE format.

Structures:

```
MathIEEE.resource:
struct MathIEEE =
{
0x00  0  struct Node MathIEEE_node;
0x0E 14  UWORD  MathIEEE_Flags
0x10 16  ULONG  MathIEEE_BaseAddr    ;für 68881-Coprozessor
0x14 20  ULONG  MathIEEE_DblBasInit  ;für andere
0x18 24  ULONG  MathIEEE_DblTransInit
0x1C 28  ULONG  MathIEEE_SnglBasInit
0x20 32  ULONG  MathIEEE_SnglTransInit
}
```

6.9 The Potgo library

The Potgo library isn't really a library. It's a resource that makes library-like calls available. The Potgo library controls the two gameports, into which you can plug mice, joysticks or paddles.

Potgo library functions

AllocPotBits — Allocates Potgo register bits

Syntax:

```
Reserved = AllocPotBits(Bits)
   D0            -6      D0
LONG Reserved;
LONG Bits;
```

Description: This function reserves bits in the Potgo register. You must request access to this register ($DFF034) before accessing the register because of the Amiga's multitasking capability. You can reserve any bits, but after the function call you must examine the bits to see if you can reserve them at that time.

Parameters:

Bits:	Bitmask that indicates which bits you want reserved. The following bits have the following meanings:
START	(Bit 0) Starts the counter for analog entries. You must set all of the entries that you want started in the same call where the OUTxx bits must be cleared.
DATLX	(Bit 8) Left port, pin 5.
OUTLX	(Bit 9) Output flag for left port, pin 5. If another task calls later with a set START bit, this port is unaffected.
DATLY:	(Bit 10) Left port, pin 9.
OUTLY:	(Bit 11) Output flag for left port, pin 9.
DATRX:	(Bit 12) Right port, pin 5.
OUTRX:	(Bit 13) Output flag for right port, pin 5.
DATRY:	(Bit 14) Right port, pin 9.
OUTRY:	(Bit 15) Output flag for right port, pin 9.

Result:

Reserved:	Sets reservable bits. This affects only the START and DATxx bits because the OUTxx bits have nothing to do with the reservation.

Comments: When you have reserved bits of the Potgo register with this function, you must free these again with `FreePotBits` once you no longer need them, or the bits will be suppressed until the next reset.

See Also: `FreePotBits`

FreePotBits — Frees Potgo register bits

Syntax:

```
FreePotBits (Reserved)
    -12          D0
LONG Reserved;
```

Description: This function frees the bits previously reserved by `AllocPotBits`.

Parameter: Reserved: The result received from `AllocPotBits`.

See Also: `AllocPotBits`

WritePotGo — Writes to Potgo register

Syntax:

```
WritePotGo (Word,Mask)
    -18        D0   D1
LONG Word, Mask;
```

Description: This function writes the specified bits in the Potgo register. It only changes the bits that are set in the mask. You should only specify bits which are actually reserved.

Parameters:

Word: New data for the Potgo register.

Mask: Bits whose values are overwritten by the bits from `Word`.

6.10 The Translator library

The Translator library converts an English sentence into a phoneme string which can then be output using the Narrator device.

Translate **Translates English into phonemes**

Syntax:

```
Error = Translate(Sentence,Length,Buffer,BufLength)
  D0       -30       A0       D0     A1       D1
LONG Error;
UBYTE *Sentence;
LONG Length;
UBYTE *Buffer;
LONG BufLength;
```

Description: This function converts an English sentence into phonemes, which the Narrator device converts into sound.

Parameters:

Sentence: Pointer to the sentence.
Length: Length of the sentence.
Buffer: Pointer to the buffer in which the phoneme codes are stored.
buflength: Length of `Buffer`.

Result: Returns a zero if no error is encountered, or a negative value which represents a negative offset in the sentence and designates the character that cannot be translated. The only error that can occur is insufficient memory in the buffer.

6.11 The Expansion library

The Expansion library makes hardware expansions available from the operating system. The following include files (Version 1.2 and up) contain additional Expansion library data:

```
<libraries/expansion.h>
<libraries/configregs.h>
<libraries/configvars.h>
<libraries/filehandler.h>
```

Some include files may not contain comments, since the manufacturer may have stripped the comments to save memory. Only non-stripped include files will contain commentary.

Expansion library functions

AddConfigDev	**Adds new ConfigDev structure**

Syntax:
```
AddConfigDev(configDev)
     -30        A0
struct ConfigDev *configDev;
```

Description: This function inserts the `ConfidDev` structure in the system's Configuration device list.

Parameter: configDev: Pointer to the `ConfigDev` structure to be inserted.

See Also: `RemConfigDev`

AddDosNode	**Adds drive to system**

Syntax:
```
ok = AddDosNode(bootPri,flags,deviceNode)
D0        -150      D0    D1     A0
BOOL ok;
BYTE bootPri;
LONG flags;
struct DeviceNode *deviceNode;
```

Description: This function adds a disk drive driver to the system. If DOS was already initialized, the function adds the driver immediately. Otherwise, the system initializes DOS then adds the driver.

Parameter:

bootpri:	Priority of the BOOT operation. It is possible to boot from any drive. The system attempts to boot from the drive assigned the highest priority. This continues until a bootable drive can be found. This allows booting from a hard disk (Kickstart 1.3 up). The following priorities are possible:
+5	Drive DF0 (usually default booting priority).
0	Hard disk.
-5	Drive in a network.
-128	Non-bootable drive.
flags:	Flags to indicate driver status. If bit 0 (`ADNF_STARTPROC`) of this flag is set, the driver starts immediately; otherwise it starts the first time the driver can be accessed. If the `dn_Taks` array in the `DeviceNode` structure is unequal to zero, this bit has no meaning.
deviceNode:	Pointer to the device node.

Result: ok: Returns FALSE if an error occurs.

Comments: If `dn_Seglist`, `dn_Handler` and `dn_Taks` of the `DeviceNode` structure contain a value of zero, the standard disk driver is used for the drive.

Warning: Because the Aztec C compiler expects long values on the stack, you must convert `bootpri` to a long value.

The task to which the device node belongs is a DOS task and not an `Exec` task. A DOS task is similar to a process, like it can create from the DOS functions.

Example: The following lines connect a bootable drive in the system and start the driver process immediately:

```
AddDosNode(0,ADNF_STARTPROC,MakeDosNode(paramPacket));
```

See Also: `MakeDosNode`

AllocBoardMem — Allocates expansion memory

Syntax:

```
startSlot = AllocBoardMem(slotSpec)
   D0           -42           D0
LONG startSlot;
LONG slotSpec;
```

Description: This function allocates sufficient expansion memory for the expansion.

Parameter: slotSpec: Expansion card memory array of type `byte`.

Result: startSlot: Number of the first usable slot.

Exceptions: Returns a value of -1 if no memory could be allocated.

Comments: Assembly language programmers should use the `EC_MEMADDR` macro to specify the memory address of the startSlot parameter.

Example:

```
struct ExpansionRom *er;
slot = AllocBoardMem(er->er_Type & ERT_MEMMASK);
```

See Also: `AllocExpansionMem`, `FreeExpansionMem`, `FreeBoardMem`

AllocConfigDev — Allocates ConfigDev structure

Syntax:

```
configDev = AllocConfigDev()
    D0           -48
struct ConfigDev *configDev;
```

Description: This function allocates a cleared `ConfigDev` structure.

Result: configDev: Pointer to a `ConfigDev` structure, or a value of zero of an error occurs

See Also: `FreeConfigDev`

AllocExpansionMem **Allocates memory for expansion**

Syntax:

```
startSlot = AllocExpansionMem(numSlots,slotOffset)
     D0              -54          D0          D1
APTR ptr;
LONG startSlot;
LONG numSlots,slotOffset;
```

Description: This function allocates memory for specific slots in expansion memory. Each slot is `E_SLOTSIZE` bytes long. After the call the following equation is fulfilled:

```
(startSlot - slotOffset) MOD slotAlign = 0
```

Parameters:

numSlots: Number of slots that should be allocated.
slotOffset: Offset of the starting slot.

Result:

startSlot: Returns the number of the starting slot, or a value of -1 if the slots could not be allocated.

Comments: Assembly language programmers should use the `EC_MEMADDR` macro to specify the memory address of the startSlot parameter.

Example:

```
AllocExpansionMem(2,0);
```

Gets two slots, where the number of the startSlot parameter must be an even address.

```
AllocExpansionMem(64,32);
```

This is the standard call for 4 megabyte memory expansion. It uses 4 megabytes of expansion memory, which ends at an odd 2 megabyte limit.

See Also: `FreeExpansionMem`, `AllocBoardMem`, `FreeBoardMem`

ConfigBoard **Configures expansion**

Syntax:

```
Error = ConfigBoard(board,configDev)
  D0         -60       A0      A1
LONG Error;
LONG board;
struct ConfigDev *configDev;
```

Description: This function configures the expansion card. This card is usually at the address E_EXPANSIONBASE, which can be changed on later model Amigas so that the address can be given as a parameter. `ConfigBoard` uses expansion memory, places the card there, and moves the `ConfidDev` structure to the newest location.

Parameters: board: Current card address.
configDev: Pointer to `ConfigDev` structure of the card.

Result: error: Returns a value other than zero if an error occurs.

See Also: `FreeConfigDev`

ConfigChain — Configures entire system

Syntax:

```
Error = ConfigChain(baseAdr)
  D0      -66          A0
LONG Error;
LONG baseAdr;
```

Description: This function configures the entire system by linking all of the expansions found at the address `E_EXPANSIONBASE` in the configuration list, and calling all the necessary functions.

Parameter: baseAdr: Base address at which the expansion card should search

Result: error: Returns a value other than zero if an error occurs

See Also: `FreeConfigDev`

FindConfigDev — Searches for ConfigDev

Syntax:

```
configDev = FindConfigDev(oldConfigDev,manufacturer,product)
     D0         -72           A0             D0           D1
struct ConfigDev *configDev;
struct ConfigDev *oldConfigDev;
LONG manufacturer,product;
```

Description: This function searches for a `ConfigDev` structure that meets the given specifications.

Parameters: oldConfigDev:
`ConfigDev` search process. If this parameter equals zero, the `ConfigDev` structure should be searched for from the beginning; otherwise it should search from the element following `oldConfigDev` structure in the list.
manufacturer: Manufacturer ID that should be searched for, or -1 if no particular manufacturer should be searched for.
product: Product ID that should be searched for, or -1 if no particular product should be searched for.

Result: configDev: Returns the pointer to the `ConfigDev` that fulfills the requirements, or a value of 0 if no `ConfigDev` structure can meet the given requirements.

Example: The following program section searches for all of the `ConfigDevs` of the list that fulfills a certain requirement:

```
struct ConfigDev *cd = Null;
while (cd = FindConfigDev(cd,MANUFACTURER,PRODUCT) )
{
   /* Here you can experiment somewhat with  */
   /* the ConfigDev */
}
```

FreeBoardMem — Frees expansion memory

Syntax:

```
FreeBoardMem(startSlot,slotSpec)
      -78        D0        D1
LONG startSlot,slotSpec;
```

Description: This function frees the memory from the expansion board previously allocated using the `AllocBoardMem` function.

Parameters:

startSlot: Number of the slot.
slotSpec: Expansion card memory array of type `byte`.

Warning: The system crashes if you attempt to free an already-free slot.

Example:

```
struct ExpansionRom *er;
LONG startSlot,slotSpec;
slotSpec = er->er_Type & ERT_MEMMASK;
startSlot = AllocBoardMem(slotSpec);
if (startSlot != -1)
{
   .
   FreeBoardMem(startSlot,slotSpec);
}
```

See Also: `AllocExpansionMem`, `FreeExpansionMem`, `AllocBoardMem`

FreeConfigDev — Frees ConfigDev structure

Syntax:

```
FreeConfigDev(configDev)
      -84        A0
struct ConfigDev *configDev;
```

Description: This function frees the `ConfigDev` structure previously allocated using the `AllocConfigDev` function.

Parameters:

congifDev: Pointer to `ConfigDev` structure.

See Also: `AllocConfigDev`

FreeExpansionMem	Frees expansion memory

Syntax:

```
FreeExpansionMem (startSlot,numSlots)
       -90          D0        D1
LONG startSlot,numSlots;
```

Description: This function frees expansion memory previously allocated using the `AllocExpansionMem` function.

Parameters:

startSlot: Starting slot.
numSlots: Number of slots.

Warning: The system crashes if you attempt to free an already-free slot.

See Also: `AllocExpansionMem`

GetCurrentBinding	Configuration parameters

Syntax:

```
number = GetCurrentBinding(currentBinding, size)
   D0          -138            A0          D0
UWORD number;
struct CurrentBinding *currentBinding;
UWORD size;
```

Description: This function gets the current parameters of the configuration.

Parameters:

currentBinding: Pointer to the `CurrentBinding` structure.
size: Size of the `CurrentBinding` structure (this size can vary).

Result: number: Returns the number of bytes copied.

See Also: `SetCurrentBinding`

MakeDosNode	Creates DOS data structures for disk

Syntax:

```
deviceNode = MakeDosNode(parameterPkt)
    D0          -144         A0
struct DeviceNode *deviceNode;
LONG *parameterPkt;
```

Description: This function creates all of the data structures necessary to add a drive to the system. These structures consist of a `DeviceNode`, a disk environment vector, a `FileSysStartupMsg` and up to two BCPL strings. The include files `<libraries/dosextens.h>` and `<libraries/filehandler.h>` contain further information. `MakesDosNode` allocates the necessary memory and links the different structures together.

Parameters: parameterPkt:

Parameter block containing the information needed for creating structures. The parameter block consists of long words of variable length:
0 = Pointer to string with the name of the DOS driver.
4 = Pointer to string with the name of the Exec driver.
8 = Number of the unit for `OpenDevice`.
12 = Flags for `OpenDevice`.
16 = Number of long words that follow.
20 = Additional information.

Result: deviceNode: Returns a pointer to the `DeviceNode` structure, or a value of zero if insufficient memory exists.

Example: The following program section adds a 3.5" disk drive to the system as DF1:

```
char execName[] = "trackdisk.device";
char dosName[]  = "df1";
ULONG parmPkt[] =
{
   (ULONG) dosName,
   (ULONG) execName,
   1,                    /* Drive number */
   0,                    /* Flags for OpenDevice */
   /* now the additional information follows */
   11,                   /* 11 additional long words */
   512 >> 2,             /* long words per block */
   0,                    /* Sector Start, unused */
   2,                    /* Number of sides */
   1,          /* Sectors per log. block, unused */
   11,                   /* Sectors per track */
   2,                    /* Reserved (boot-) blocks */
   0,                    /* Unused */
   0,                    /* Interleave factor */
   0,                    /* First cylinder */
   79,                   /* Last cylinder */
   5                     /* Number of the buffer */
};
struct DeviceNode *dnode,*MakeDosNode();
dnode = MakeDosNode(parmPkt);
```

See Also: `AddDosNode`

ObtainConfigBinding — Gets ConfigDev access rights

Syntax:

```
ObtainConfigBinding ()
        -120
```

Description: This function must be called before adding a driver to the `ConfigDev` structure, if you wish to add the driver using means other than the CLI `BindDrivers` command. The `ObtainConfigBinding` function ensures that no other programs want to add drivers at the same time.

See Also: `ReleaseConfigBinding`

ReadExpansionByte — Reads byte nibble by nibble

Syntax:

```
Byte = ReadExpansionByte(Board,Offset)
 D0           -96          A0    D0
BYTE Byte;
LONG Board,Offset;
```

Description: This function reads a byte from a new expansion card. New expansion card data is read from memory nibble by nibble.

Parameters:

Board: Pointer to base address of an expansion card.
Offset: Offset of expansion ROM structure, calculated using the `EROFFSET` and `ECOFFSET` macros.

Result:

Byte: Returns the byte that was read, or a value of -1 if the byte could not be read.

Comments: The `ReadExpansionRam` function usually calls this function.

Example:

```
type = ReadExpansionByte(cd->BoardAddr,EROFFSET(er_Type));
ints =
   ReadExpansionByte(cd->BoardAddr,ECOFFSET(ec_Interrupt));
```

See Also: `ReadExpansionRom`, `WriteExpansionByte`

ReadExpansionRom — Read configuration data

Syntax:

```
Error = ReadExpansionRom(board,configDev)
  D0            -102          A0    A1
LONG Error;
LONG board;
struct ConfigDev *configDev;
```

Description: This function reads the configuration data of an expansion card in the `ConfigDev` structure (`cd_Rom`). The function determines whether a card exists in the given address, as well as whether the card is a new expansion card or an old expansion card.

Parameters:

board: Pointer to base address of an expansion card.
configDev: Pointer to `ConfigDev` structure.

Result:

error: Returns a value other than zero if an error occurs.

Example:
```
configDev = AllocConfigDev();
if (!configDev) Error();
Error = ReadExpansionRom(board,configDev);
if (!Error)
{
   configDev->cd_BoardAddr = board;
   ConfigBoard(configDev);
}
```

See Also: ReadExpansionByte, WriteExpansionByte

ReleaseConfigBinding — Frees access rights

Syntax:
```
ReleaseConfigBinding ()
      -126
```

Description: Releases access rights established by the ObtainConfigBinding function, so that other programs can add their drivers.

See Also: ObtainConfigBinding

RemConfigDev — Removes ConfigDev structure

Syntax:
```
RemConfigDev(configDev)
     -108       A0
struct ConfigDev *configDev;
```

Description: This function removes the given ConfigDev structure from the list of all of the ConfigDev structures.

Parameter: configDev: Pointer to ConfigDev structure.

See Also: AddConfigDev

SetCurrentBinding — Sets configuration parameters

Syntax:
```
SetCurrentBinding (currentBinding,number)
       -132           A0            D0
struct CurrentBinding *currentBinding;
UWORD number;
```

Description: This function specifies the current configuration parameters, allowing the option of adding parameters to an existing device.

Parameters:

currentBinding: Pointer to CurrentBinding structure.

number: Number of bytes that should be given.

See Also: GetCurrentBinding

WriteExpansionByte — Writes a byte by nibbles

Syntax:

```
Error = WriteExpansionByte(board,offset,byte)
  D0              -114        A0      D0   D1
LONG Error;
LONG board,offset;
BYTE byte;
```

Description: This function writes a byte to a new expansion card. New expansion card data is written to memory nibble by nibble.

Parameters:

Board: Pointer to base address of an expansion card.

Offset: Offset of expansion ROM structure, calculated using the `EROFFSET` and `ECOFFSET` macros.

Result:

Byte: Returns the byte that was read, or a value of -1 if the byte could not be read.

Warning: Because the Aztec C compiler requires long values on the stack, you must convert the byte to a long word.

Example:

```
Error = WriteExpansionByt
            (cd->BoardAddr,ECOFFSET(ec_Shutup),(LONG)0);
Error = WriteExpansionByte
            (cd->BoardAddr,ECOFFSET(ec_Interrupt),1L);
```

See Also: `ReadExpansionByte`, `ReadExpansionRom`

Structures:

Global sizes:

```
E_SLOTSIZE              0x10000
E_EXPANSIONBASE         0xe80000
E_EXPANSIONSIZE         0x080000
E_EXPANSIONSLOTS        8
E_MEMORYBASE            0x200000
E_MEMORYSIZE            0x800000
E_MEMORYSLOTS           128
```

```
struct ExpansionRom <libraries/configregs.h>
{
0x00  0  UBYTE       er_Type;
0x01  1  UBYTE       er_Product;
0x02  2  UBYTE       er_Flags;
0x03  3  UBYTE       er_Reserved03;
0x04  4  UWORD       er_Manufacturer;
0x06  6  ULONG       er_SerialNumber;
0x0A 10  UWORD       er_InitDiagVec;
0x0C 12  UBYTE       er_Reserved0c;
0x0D 13  UBYTE       er_Reserved0d;
0x0E 14  UBYTE       er_Reserved0e;
0x0F 15  UBYTE       er_Reserved0f;
0x10 16
};
```

Types (er_Type):

```
ERT_TYPEMASK            0xc0
ERT_NEWBOARD            0xc0
ERT_MEMMASK             0x07
ERTF_CHAINEDCONFIG      (1<<3)
ERTF_DIAGVALID          (1<<4)
ERTF_MEMLIST            (1<<5)
Flags (er_Flags):
ERFF_MEMSPACE           (1<<7)
ERFF_NOSHUTUP           (1<<6)

struct ExpansionControl <libraries/configregs.h>
{
0x00  0  UBYTE       ec_Interrupt;
0x01  1  UBYTE       ec_Reserved11;
0x02  2  UBYTE       ec_BaseAddress;
0x03  3  UBYTE       ec_Shutup;
0x04  4  UBYTE       ec_Reserved14;
0x05  5  UBYTE       ec_Reserved15;
0x06  6  UBYTE       ec_Reserved16;
0x07  7  UBYTE       ec_Reserved17;
0x08  8  UBYTE       ec_Reserved18;
0x09  9  UBYTE       ec_Reserved19;
0x0A 10  UBYTE       ec_Reserved1a;
0x0B 11  UBYTE       ec_Reserved1b;
0x0C 12  UBYTE       ec_Reserved1c;
0x0D 13  UBYTE       ec_Reserved1d;
0x0E 14  UBYTE       ec_Reserved1e;
0x0F 15  UBYTE       ec_Reserved1f;
0x10 16
};
```

Interrupt Control register (ec_Interrupt):

```
ECIF_INTENA             (1<<1)
ECIF_RESET              (1<<3)
ECIF_INT2PEND           (1<<4)
ECIF_INT6PEND           (1<<5)
ECIF_INT7PEND           (1<<6)
ECIF_INTERRUPTING       (1<<7)

struct ConfigDev <libraries/configvars.h>
{
0x00  0  struct Node         cd_Node;
0x0E 14  UBYTE               cd_Flags;
0x0F 15  UBYTE               cd_Pad;
0x10 16  struct ExpansionRom cd_Rom;
0x20 32  APTR                cd_BoardAddr;
0x24 36  APTR                cd_BoardSize;
0x28 40  UWORD               cd_SlotAddr;
0x2A 42  UWORD               cd_SlotSize;
0x2C 44  APTR                cd_Driver;
0x30 48  struct ConfigDev *  cd_NextCD;
0x34 52  ULONG               cd_Unused[4];
0x44 68
};
```

Flags (cd_Flags):

```
CDF_SHUTUP      0x01
CDF_CONFIGME    0x02

struct CurrentBinding <libraries/configvars.h>
{
0x00  0  struct ConfigDev *cb_ConfigDev;
0x04  4  UBYTE             *cb_FileName;
0x08  8  UBYTE             *cb_ProductString;
0x0C 12  UBYTE            **cb_ToolTypes;
0x10 16
};

struct FileSysStartupMsg <libraries/filehandler.h>
{
0x00  0  ULONG       fssm_Unit;
0x04  4  BSTR        fssm_Device;
0x08  8  BPTR        fssm_Environ;
0x0C 12  ULONG       fssm_Flags;
0x10 16
};

struct DeviceNode <libraries/filehandler.h>
{
0x00  0  BPTR            dn_Next;
0x04  4  ULONG           dn_Type;
0x08  8  struct MsgPort *dn_Task;
0x0C 12  BPTR            dn_Lock;
0x10 16  BSTR            dn_Handler;
0x14 20  ULONG           dn_StackSize;
0x18 24  LONG            dn_Priority;
0x1C 28  BPTR            dn_Startup;
0x20 32  BPTR            dn_SegList;
0x24 36  BPTR            dn_GlobalVec;
0x28 40  BSTR            dn_Name;
0x2C 44
};
```

6.12 The RomBoot library

The RomBoot library allows Kickstart 1.3 to boot from drive DF0:, or from any expansion card added to the Amiga. The expansion card must be bootable and have a ROM which contains the necessary initialization routine.

When booting, the operating system tries to boot as usual from DF0:. If no disk is in this drive, or if the disk is non-bootable, the system checks all of the expansion cards to see if they are bootable and then boots the card with the highest priority (see Expansion library: `AddDosNode`). If no bootable expansion cards exist, the icon appears on the screen requesting that you insert a Workbench disk.

Expansion cards could be bootable from the hard disk driver, and networked Amigas can also boot from the network.

The RomBoot library contains only functions that are currently undocumented by Commodore-Amiga. Because this function should only be used within the operating system when booting, the average user or programmer may not find this information useful.

Structures:

```
struct RomBootBase
{
0x00  0  struct Library   LibNode;
0x22 34  struct Execbase *ExecBase;
0x26 38  struct List      BootList;
0x34 52  ULONG            Reserved[4];
0x38 56
};

struct BootNode
{
0x00  0  struct Node bn_Node;
0x0E 14  UWORD       bn_Flags;
0x10 16  CPTR        bn_DeviceNode;
0x14 20
};
```

6.13 The Console library

The Console library isn't really a library, but it only handles two console device functions which are called as library functions. A pointer to the console library is needed, such as this:

```
if (OpenDevice("console.device",-1L,IOStdReq,0L) == 0)
   ConsoleDevice = IOStdReq->io_Device;
else
   /* error */
```

Console library functions

CDInputHandler — Sends input to Console device

Syntax:

```
CDInputHandler (events,ConsoleDevice)
      -42          A0        A1
struct Events *events;
struct Device *ConsoleDevice;
```

Description: This function receives input (normally from the `input.task` ROM) and sends this information to the console device.

Parameters:

events: Pointer to list of events.
ConsoleDevice:
Pointer to console device.

Comments: This function is listed here for historical purposes only, and should not be used. Input in the system is handled by the `WriteEvent` function of the input device.

RawKeyConvert — Converts RAWKEY to ASCII

Syntax:

```
number = RawKeyConvert(event,buffer,length,keyMap)
 D0          -48          A0     A1     D1     A2
SHORT number;
struct InputEvent *event;
UBYTE *buffer;
LONG length;
struct KeyMap *keyMap;
```

Description: This function converts RAWKEY events into ASCII characters, or into an ANSI character string. This happens in conjunction with the `KeyMap`, if one exists.

Parameters:	event:	Pointer to input event.
	buffer:	Pointer to data buffer containing the ANSI character string.
	length:	Data buffer size in bytes.
	keyMap:	Pointer to `KeyMap` which converts the RAWKEYs into ANSI character strings. If `keyMap` equals zero the default `KeyMap` is used.
Result:	number:	Returns the number of characters written in the data buffer, or a value of -1 if insufficient buffer memory existed.

Exceptions: If the `number` parameter equals -1, the contents of the data buffer are declared invalid.

Structures:

```
struct InputEvent <devices/inputevent.h>
{
0x00  0  struct  InputEvent *ie_NextEvent;
0x04  4  UBYTE   ie_Class;           /* Type */
0x05  5  UBYTE   ie_SubClass;        /* Sub type */
0x06  6  UWORD   ie_Code;
0x08  8  UWORD   ie_Qualifier;
0x0A 10  union
         {
            struct
            {
0x0A 10        WORD    ie_x;         /* Mouse position */
0x0C 12        WORD    ie_y;
            } ie_xy;
0x0A 10     APTR    ie_addr;
         } ie_position;
0x0E 14  struct timeval ie_TimeStamp;/* System time */
0x16 22
};
```

Event classes (ie_Class):

```
IECLASS_Null             0x00
IECLASS_RAWKEY           0x01
IECLASS_RAWMOUSE         0x02
IECLASS_EVENT            0x03
IECLASS_POINTERPOS       0x04
IECLASS_TIMER            0x06
IECLASS_GADGETDOWN       0x07
IECLASS_GADGETUP         0x08
IECLASS_REQUESTER        0x09
IECLASS_MENULIST         0x0A
IECLASS_CLOSEWINDOW      0x0B
IECLASS_SIZEWINDOW       0x0C
IECLASS_REFRESHWINDOW    0x0D
IECLASS_NEWPREFS         0x0E
IECLASS_DISKREMOVED      0x0F
IECLASS_DISKINSERTED     0x10
IECLASS_ACTIVEWINDOW     0x11
IECLASS_INACTIVEWINDOW   0x12
```

Event Codes (ie_Code):

RAWKEY-Codes:

```
IECODE_UP_PREFIX         0x80
IECODE_KEY_CODE_FIRST    0x00
IECODE_KEY_CODE_LAST     0x77
IECODE_COMM_CODE_FIRST   0x78
IECODE_COMM_CODE_LAST    0x7F
```

ANSI Codes:

```
IECODE_C0_FIRST          0x00
IECODE_C0_LAST           0x1F
IECODE_ASCII_FIRST       0x20
IECODE_ASCII_LAST        0x7E
IECODE_ASCII_DEL         0x7F
IECODE_C1_FIRST          0x80
IECODE_C1_LAST           0x9F
IECODE_LATIN1_FIRST      0xA0
IECODE_LATIN1_LAST       0xFF
```

RAWMOUSE Codes:

```
IECODE_LBUTTON           0x68
IECODE_RBUTTON           0x69
IECODE_MBUTTON           0x6A
IECODE_NOBUTTON          0xFF
```

WINDOW Codes:

```
IECODE_NEWACTIVE         0x01
```

REQUESTER Codes:

```
IECODE_REQSET            0x01
IECODE_REQCLEAR          0x00
```

Event Qualifier (ie_Qualifier):

```
IEQUALIFIER_LSHIFT          0x0001
IEQUALIFIER_RSHIFT          0x0002
IEQUALIFIER_CAPSLOCK        0x0004
IEQUALIFIER_CONTROL         0x0008
IEQUALIFIER_LALT            0x0010
IEQUALIFIER_RALT            0x0020
IEQUALIFIER_LCOMMAND        0x0040
IEQUALIFIER_RCOMMAND        0x0080
IEQUALIFIER_NUMERICPAD      0x0100
IEQUALIFIER_REPEAT          0x0200
IEQUALIFIER_INTERRUPT       0x0400
IEQUALIFIER_MULTIBROADCAST  0x0800
IEQUALIFIER_MIDBUTTON       0x1000
IEQUALIFIER_RBUTTON         0x2000
IEQUALIFIER_LEFTBUTTON      0x4000
IEQUALIFIER_RELATIVEMOUSE   0x8000
```

```
struct KeyMap <devices/keymap.h>
{
0x00  0  UBYTE   *km_LoKeyMapTypes;
0x04  4  ULONG   *km_LoKeyMap;
0x08  8  UBYTE   *km_LoCapsable;
0x0C 12  UBYTE   *km_LoRepeatable;
0x10 16  UBYTE   *km_HiKeyMapTypes;
0x14 20  ULONG   *km_HiKeyMap;
0x18 24  UBYTE   *km_HiCapsable;
0x1C 28  UBYTE   *km_HiRepeatable;
0x20 32
};
```

Keymap Types:

```
KC_NOQUAL   0
KC_VANILLA  7
KCF_SHIFT   0x01
KCF_ALT     0x02
KCF_CONTROL 0x04
KCF_DOWNUP  0x08
KCF_DEAD    0x20
KCF_STRING  0x40
KCF_NOP     0x80
```

Prefix Codes for DEAD Keys:

```
DPF_MOD          0x01    ; Key was modified by dead key
DPF_DEAD         0x08    ; Dead Key
DP_2DINDEXMASK   0x0f
DP_2DFACSHIFT    4
```

6.14 The Timer library

The Timer library makes three library-like functions available for time control. This is helpful when your program involves operations requiring time.

Timer library functions

AddTime — Adds two times

Syntax:

```
AddTime (Dest,Source)
  -42     A0    A1
struct timeval *Dest,*Source;
```

Description: This function adds the two times specified by the `timeval` structures. The result is placed in the `Dest timeval` structure.

Parameters:

Dest: Pointer to `timeval` structure which contains the result after the call.

Source: Pointer to `timeval` structure.

Comments: Registers A0 and A1 remain unchanged.

CmpTime — Compares two times

Syntax:

```
result = CmpTime(Dest,Source)
   D0      -54    A0    A1
LONG result;
struct timeval *Dest,*Source;
```

Description: This function compares two times.

Parameters:

Dest: Pointer to `timeval` structure.

Source: Pointer to `timeval` structure.

Result:

result: Returns a value of zero if both times are identical, a value of +1 if `Dest` has more time than `Source` and a value of -1 if `Dest` has less time than `Source`

Comments: Registers A0 and A1 remain unchanged.

SubTime — Subtracts two times

Syntax:

```
SubTime (Dest,Source)
   -48    A0    A1
timeval *Dest,*Source;
```

Description: This function subtracts the two times specified by the `timeval` structures. The result is saved in the `Dest timeval` structure:

```
dest = dest - source
```

Parameters:

Dest: Pointer to `timeval` structure which contains the result after the call.

Source: Pointer to `timeval` structure.

Comments: Registers A0 and A1 remain unchanged.

Structures:

```
struct timeval <devices/timer.h>
{
0x00 0  ULONG tv_secs;  /* Seconds */
0x04 4  ULONG tv_micro; /* Microseconds [0,999999] */
0x08 8
}
```

7. Basic and System Structures

Many kinds of computer data exist. There's data which you use very seldom, data you almost always use, and most often, data that you couldn't operate your computer without. For example, you don't access fonts directly very often, but the system uses fonts constantly. Without fonts of some kind, you wouldn't be able to display any text using the Amiga.

This chapter examines the basic structures of the Amiga operating system, right down to the system structures. How does the keyboard check operate without keymaps? Or how do you print something without a printer driver? It doesn't, and you don't, without these system structures, set by the operating system or by applications.

Remember that the Amiga is an ever expanding system and therefore the system structures will change. Never count on the addresses of system structures to remain constant between versions of the operating system and never modify private system structures.

7.1 Preferences as a data structure

Every computer includes basic settings that define its "character." In the early days of home computing, the manufacturer dictated these settings. For example, you could be sure that when you turned on a Commodore 64, the screen and text would appear in different shades of blue.

With the accent on user-friendliness, you can now change these settings to suit your own needs. This makes it much more enjoyable to work with your machine.

The Amiga has so many settings that it includes an extra structure for storing these settings. If a boot disk doesn't contain this information, the Amiga keeps a default `Preferences` structure in Kickstart ROM.

7.1.1 The data managed by Preferences

The `Preferences` structure, which is managed from Intuition, contains all of the data that can be set using the Preferences program. The `Preferences` structure looks like this:

```
struct Preferences
{
0x000 000  BYTE    FontHeight;
0x001 001  UBYTE   PrinterPort;
0x002 002  USHORT  BaudRate;
0x004 004  struct  timeval KeyRptSpeed;
0x004 004   ULONG  tv_secs;
0x008 008   ULONG  tv_micro;
0x00C 012  struct  timeval KeyRptDelay;
0x00C 012   ULONG  tv_secs;
0x010 016   ULONG  tv_micro;
0x014 020  struct  timeval DoubleClick;
0x014 020   ULONG  tv_secs;
0x018 024   ULONG  tv_micro;
0x01C 028  USHORT  PointerMatrix[36];
0x064 100  BYTE    XOffset;
0x065 101  BYTE    YOffset;
0x066 102  USHORT  color17;
0x068 104  USHORT  color18;
0x06A 106  USHORT  color19;
0x06C 108  USHORT  PointerTicks;
0x06E 110  USHORT  color0;
0x070 112  USHORT  color1;
0x072 114  USHORT  color2;
0x074 116  USHORT  color3;
0x076 118  BYTE    ViewXOffset;
0x077 119  BYTE    ViewYOffset;
0x078 120  WORD    ViewInitX;
0x07A 122  WORD    ViewInitY;
0x07C 124  BOOL    EnableCLI;
0x07D 126  USHORT  PrinterType;
0x080 128  UBYTE   PrinterFilename[FILENAME_SIZE];
0x09E 158  USHORT  PrintPitch;
0x0A0 160  USHORT  PrintQuality;
0x0A2 162  USHORT  PrintSpacing;
0x0A4 164  UWORD   PrintLeftMargin;
0x0A6 166  UWORD   PrintRightMargin;
0x0A8 168  USHORT  PrintImage;
0x0AA 170  USHORT  PrintAspect;
0x0AC 172  USHORT  PrintShade;
0x0AE 174  WORD    PrintThreshold;
0x0B0 176  USHORT  PaperSize;
0x0B2 178  UWORD   PaperLength;
0x0B4 180  USHORT  PaperType;          /* End Ver. 1.1 */
0x0B6 182  UBYTE   SerRWBits;
0x0B7 183  UBYTE   SerStopBuf;
0x0B8 184  UBYTE   SerParShk;
0x0B9 185  UBYTE   LaceWB;
0x0BA 186  UBYTE   WorkName[30];
0x0BB 187  BYTE    RowSizeChange;
```

```
0x0BC 188  BYTE    ColumnSizeChange;   /* End Ver. 1.2 */
0x0BE 190  UWORD   PrintFlags;
0x0C0 192  UWORD   PrintMaxWidth;
0x0C2 194  UWORD   PrintMaxHeight;
0x0C4 196  UBYTE   PrintDensity;
0x0C5 197  UBYTE   PrintXOffset;
0x0C6 198  UWORD   wb_Width;
0x0C8 200  UWORD   wb_Height;
0x0CA 202  UBYTE   wb_Depth;           /* Ver. 1.3 */
0x0CB 203  UBYTE   ext_size;
0x0CD 204
};
```

The structure can be divided into the subject groups listed below.

Time intervals

All of the system time intervals applying to the keyboard and the mouse can be set. This means that the `timeval` structures are integrated into the `Preferences` structure. They use components made of seconds and microseconds. The `keyRptSpeed` field specifies the frequency at which the key repeats. The smaller this time value, the more characters are repeated on the screen at a time.

Before a key repeats, the keyboard driver delays repetition until a certain interval elapses. This interval is called the *repetition threshold*, which is defined by the `KeyRptDelay` structure. The larger the value, the longer the system waits until beginning key repeat.

The `DoubleClick` structure specifies the time interval that elapses between the two clicks of a double-click. This structure indicates the maximum amount of time between two single clicks before they may be read as a double-click. If you set the `DoubleClick` value too high, the Workbench may read closely spaced single clicks as double-clicks.

Mouse pointer

The `Preferences` structure includes a `Pointermatrix` field, which describes the mouse pointer graphic data. It is stored in an array of 36 words, which gives us a possible 16x16 pixel, four color pointer. This array remains constant—we cannot expand the pointer to a larger size.

`XOffset` and `YOffset` variables contain the coordinate values of the mouse's hot spot (the pixel of the mouse pointer where the actual activation occurs).

The `color17`, `color18` and `color19` fields control three colors of the mouse pointer sprite, based on color registers 17 through 19.

The `PointerTicks` value defines the number of ticks needed to move the mouse by one increment. Normal settings are limited to 1, 2 and 4. The higher the number, the farther you would have to move the mouse to move the pointer one increment.

Workbench

Our basic structures also store some values which control the Workbench's appearance.

The `color0`, `color1`, `color2` and `color3` fields specify the Workbench colors.

The `ViewXOffset` and `ViewYOffset` fields place the upper left corner of the Workbench screen (the `View`) at its proper location of the monitor screen (you adjust this parameter from the center screen of Preferences).

The `ViewInitX` and `ViewInitY` fields contain the initial View values.

The `FontHeight` entry specifies the height of the default Topaz font (8/9).

Version 1.3 includes additional fields: `wb_Depth`, `wb_Height` and `wb_width`. They contain the dimensions and depth of the Workbench screen. They can be read from any programs that open a window on the screen.

The `EnableCLI` field (Versions 1.1-1.2) determined whether the CLI icon appeared on the Workbench screen. Versions 1.3 and up ignore this setting.

Printer settings

Almost half the `Preferences` structure lists printer parameters. This is understandable when you consider how many settings many printers offer.

The first and most elementary setting is the `PrinterPort` field, which indicates the port through which the printer is driven. This is set to `PARALLEL_PRINTER` or `SERIAL_PRINTER`.

Commodore-Amiga provides its own printer types that are then supported through a printer driver. The `PrinterType` field reads available printer types. If the printer cannot be found, `CustomPrinter` is chosen and the name of the printer or printer driver is placed in the PrinterFileName field. The system can be expanded at any time.

The PrintPitch field specifies the pitch (characters per line). Normal (pica) type prints 10 CPI (characters per inch). Elite pitch prints 12 CPI, and Fine (also called condensed) pitch prints 15 CPI.

The PrintQuality field controls the printed quality of the text. Many modern printers support the Near Letter Quality (NLQ) or LQ in addition to draft quality.

The PrintSpacing field indicates the number of printed lines per inch on a page. Here you can choose between 6 LPI (lines per inch) and 8 LPI.

Now the printer needs to know the dimensions of the printed page. The PrintLeftmargin and PrintRightMargin fields give the right and left margins of the paper.

The PaperSize field indicates the size of a sheet of paper. If you're using some unusual paper size, you can insert the total number of lines that will fit on a printed page in the PaperLength field instead.

The PaperType field can indicate either single sheets of paper or fanfold (continuous) paper.

Graphic printout settings

The Amiga as a graphic computer offers many different settings for graphic printing. The PrintImage field allows the option of generating a positive or negative printout. This is especially practical if a picture has many dark surfaces that will appear black with a positive print. Inverting the color causes less wear and tear on the printer ribbon, and may yield a better printout.

The PrintAspect field specifies the aspect (direction) of the print.

The PrintShade field controls the intensity of shading. Here you can specify whether the printout should be in black and white, in gray scales or in color. The PrintThreshold field controls the degree of gray scaling as dictated by PrintShade.

The expansions that came with the new operating system for setting the graphic printout are also important in the Preferences structure. Version 1.3 includes additional values that apply to the Preferences program.

PrintFlags contains various settings for fine-tuning graphic printed output. The settings for Smoothing (ON, OFF), ColorCorrect

(R, G, B), Dithering (Ordered, Halftone, F-S) and Scaling (Fraction, Integer) are stored in PrintFlags.

The PrintMaxWidth and PrintMaxHeight fields contain the topmost limits of a graphic's printed height and width.

Density is saved in an extra byte because this value will probably be improved in later versions.

The PrintXOffset field specifies the left offset of the graphic printout.

Serial data transfer

Since Version 1.3 of the operating system, additional settings for the serial interface are now saved in the Preferences structure. These are:

SerRWBit, one byte in whose top section the number of bits to be read is saved and in whose bottom section the number of bits to be written is saved. A setting of either 7 or 8 is possible.

The number of stop bits (1 or 2) as well as the size of the data buffer are saved in SerStopBuf.

SerParShk acts as a flag for two settings. You can choose between three handshake methods (xOn/xOff), RTS/CTS and None) and between three parities (none, even and odd), using the top and bottom bits as described above.

7.1.2 Preferences access through Intuition

After this information about the individual values of the data structure we now come to some examples about the practicality. The Preferences structure can be accessed through the Preferences program. However, we'd like to draw up a short scenario for you to show how impractical this can be.

An early Amiga word processor relied completely on Preferences settings, prohibiting the user from making small changes before a printout. It was incompatible with other programs.

The single advantage is the amount of memory saved by not using memory for input and modifications. Let's assume that you type in a text and would like to print this out twice: Once with normal type and once with NLQ type. The Preferences program must be loaded, which may or may not fit into available memory.

Today we know that the `Preferences` structure can be changed without actually loading Preferences, and without conflicting with the system or other programs. This is because Intuition supports three functions. The first function offers the option of transferring the contents of the current structure into buffer memory. We supply the pointer to our buffer and the number of bytes that should be copied. This `GetPrefs()` function has the following general format:

```
GetDefPrefs(PrefBuffer, Size);
     -132      A0        D0
```

If we allocate a memory region and then copy the values there, we can read the current values first. The following program does just that:

```
/***************************************
 *                                     *
 * Program: Read Preferences Data      *
 * =================================== *
 *                                     *
 * Author:  Date:      Comments:       *
 * ------  ----------  ----------      *
 * Wgb     07/03/1988  printer data    *
 *  cc pref.c                          *
 *  ln pref.o -lc32                    *
 ***************************************/
#include <exec/memory.h>
#include <intuition/intuition.h>
#define SIZE sizeof(struct Preferences)
struct IntuitionBase *IntuitionBase;
struct Preferences   *PrefsBuffer;
main()
   {
   Open_All();
   GetPrefs(PrefsBuffer, SIZE);
   printf("Printer settings:\n");
   printf("Pitch  : $%4x\n", PrefsBuffer->PrintPitch);
   printf("Quality: $%4x\n", PrefsBuffer->PrintQuality);
   printf("Spacing: $%4x\n", PrefsBuffer->PrintSpacing);

   printf("Right Margin: %d\n", PrefsBuffer->PrintRightMargin);
   printf("Left  Margin: %d\n", PrefsBuffer->PrintLeftMargin);
   Close_All();
   }
/***************************************
 * Function: Open Library & Memory     *
 * =================================== *
 *                                     *
 * Author:  Date:      Comments :      *
 * ------  ----------  ----------      *
 * Wgb     07/03/1988                  *
 *                                     *
 ***************************************/
Open_All()
   {
   void  *OpenLibrary();
```

```
   UBYTE *AllocMem();
   if (!(IntuitionBase = (struct IntuitionBase *)
       OpenLibrary("intuition.library", 0L)))
      {
      printf("No Intuition Library found!\n");
      Close_All();
      exit(FALSE);
      }
   PrefsBuffer = (struct Preferences *)AllocMem(SIZE,
   MEMF_CLEAR|MEMF_FAST);
   if (!PrefsBuffer)
      {
      printf("No more memory!\n");
      Close_All();
      exit(FALSE);
      }
   }
/***************************************
 * Function: Close anything now open    *
 * ================================== *
 * Author:  Date:       Comments:       *
 * ------  ----------  ----------      *
 * Wgb     10.16/1987  Intuition and    *
 *                     memory only      *
 ***************************************/
Close_All()
   {
   if (PrefsBuffer)     FreeMem(PrefsBuffer, SIZE);
   if (IntuitionBase)   CloseLibrary(IntuitionBase);
   }
```

Program description

The program opens the Intuition.Library and allocates a memory range in which the main program places the data of the current `Preferences` structure. The chosen values are displayed and the program closes the files.

This alone may be all that's needed for your program. Let's assume that your program runs only on an Interlace Workbench. Then at the beginning of your program you check to see if the `Preferences` structure has an entry under `LaceWB`. If not, the program stops with an error message.

Another alternative is a short-term change to the `Preferences` structure. Many programs load the structure into the buffer twice. The first copy acts as the pattern for returning to default values later. Many entries are changed in the second copy, and then the entire structure is sent as a new setting to the operating system. This is done using the `SetPrefs()` function:

```
SetPrefs(PrefBuffer, Size, Flag)
 -324        A0       D0     D1
```

We assign a pointer to the structure, our buffer range and the size of the buffer, because it may not be necessary to copy the entire structure and then return it. We include a flag value which determines whether other programs should change the structure. If this flag value is set to TRUE, all of the programs containing a NEWPREFS message are informed. A value of FALSE ignores this message.

The following program uses the `Preferences` structure exactly like the first program, changes some of the values and returns them. Because the changes are rather trivial (some colors were changed) there is no NEWPREFS message as for the other program.

```
/****************************************
 * Program: Change color data          *
 * =================================== *
 * Author:  Date:      Comments:        *
 * ------  ----------  ----------       *
 * Wgb     07/03/1988  Color data       *
 *  cc pref.c                           *
 *  ln pref.o -lc32                     *
 ****************************************/
#include <exec/memory.h>
#include <intuition/intuition.h>
#define SIZE sizeof(struct Preferences)
struct IntuitionBase *IntuitionBase;
struct Preferences   *PrefsBuffer;
main()
   {
   Open_All();
   GetPrefs(PrefsBuffer, SIZE);
   PrefsBuffer->color0 = 1*15 + 256* 15;
   PrefsBuffer->color1 = 1;
   PrefsBuffer->color2 = 16*15;
   SetPrefs(PrefsBuffer, SIZE, FALSE);
   Close_All();
   }
/****************************************
 * Function: Open Library & Memory     *
 * =================================== *
 * Author:  Date:      Comments :       *
 * ------  ----------  ----------       *
 * Wgb     07/03/1988                   *
 ****************************************/
Open_All()
   {
   void  *OpenLibrary();
   UBYTE *AllocMem();
   if (!(IntuitionBase = (struct IntuitionBase *)
       OpenLibrary("intuition.library", 0L)))
      {
      printf("No Intuition Library found!\n");
      Close_All();
      exit(FALSE);
      }
   PrefsBuffer = (struct Preferences *)AllocMem(SIZE,
   MEMF_CLEAR|MEMF_FAST);
   if (!PrefsBuffer)
      {
```

```
        printf("No more memory!\n");
        Close_All();
        exit(FALSE);
        }
    }
/****************************************
 * Function: Close anything now open    *
 * ==================================== *
 * Author:  Date:       Comments:       *
 * ------   ----------  ----------      *
 * Wgb      10/16/1987  Intuition and   *
 *                      memory only     *
 *                                      *
 ****************************************/
Close_All()
    {
    if (PrefsBuffer)     FreeMem(PrefsBuffer, SIZE);
    if (IntuitionBase)   CloseLibrary(IntuitionBase);
    }
```

7.2 Printer drivers

What do you do when you have purchased a printer that is not supported by the Amiga? This can be solved in most cases by developing your own printer driver. The printer driver mediates between the printer device and the printer.

What does a printer driver look like? First of all it needs a header, similar to the fonts. So that a printer driver doesn't disrupt the computer with a false start, the first four bytes of the driver contain the assembly language instructions:

```
moveq #0,d0
rts
```

Next follow two words that give the version and revision number of the printer driver. The following is the `PrinterExtended` data structure:

```
Offset    Structure
------    --------
          struct PrinterExtendedData
          {
0  0x00       char  *ped_PrinterName;  /* Printer name */
4  0x04       VOID  (*ped_Init)();     /* Initialization
                                          routine */
8  0x08       VOID  (*ped_Expunge)();  /* De-Initialization
                                          routine */
12 0x0c       VOID  (*ped_Open)(); /* called by OpenDevice() */
16 0x10       VOID  (*ped_Close)();    /* called by
                                          CloseDevice() */
20 0x14        UBYTE ped_PrinterClass;
21 0x15        UBYTE ped_ColorClass;
22 0x16        UBYTE ped_MaxColumns;    /* number of printer
                                           columns (e.g., 80 or
                                           136 ) */
23 0x17        UBYTE ped_NumCharSets;   /* number of  printer
                                           fonts */
24 0x18        UWORD ped_NumRows;       /* number of printer
                                           lines */
26 0x1a        ULONG ped_MaxXDots;      /* Maximum horiz.
                                           resolution */
30 0x1e        ULONG ped_MaxYDots;      /* Maximum vert.
                                           resolution */
34 0x22        UWORD ped_XDotsInch;     /* Dots per inch
                                           (horiz.) */
36 0x24        UWORD ped_YDotsInch;     /* Dots per inch
                                           (vert.) */
38 0x26        char  ***ped_Commands;   /* Command strings */
42 0x2a        VOID  (*ped_DoSpecial)();/* Special command
                                           handler */
46 0x2e        VOID  (*ped_Render)();    /* Hardcopy routine */
```

```
50 0x32        LONG  ped_TimeoutSecs;
54 0x36        char  **ped_8BitChars;
58 0x3a     }; /* defined in "devices/prtbase.h" */
```

The arrays and functions defined in this structure follow.

7.2.1 The PrinterExtendedData structure

ped_Init: This routine is called after the printer driver is loaded from the printer device. This routine gives a pointer to a PrinterData structure:

```
VOID Init (PrinterData)
struct PrinterData
          *PrinterData;
{..}
```

The PrinterData structure has the following appearance:

```
Offset      Structure
------      ---------
            struct PrinterData
            {
0     0x00     struct DeviceData      pd_Device;
52    0x34     struct MsgPort         pd_Unit;
86    0x56     BPTR          pd_PrinterSegment; /* For
                                                 UnLoadSeg() */
90    0x5a     UWORD                  pd_PrinterType;
92    0x5c     struct PrinterSegment *pd_SegmentData;
96    0x60     UBYTE                 *pd_PrintBuf;  /* Buffer */
100   0x64     int                 (*pd_PWrite)(); /* Write
                                                   function */
104   0x68     int                   (*pd_PBothReady)(); union {
108   0x6c        struct IOExtPar  pd_p0; /* Parallel */
108   0x6c        struct IOExtSer  pd_s0; /* Serial   */
               }
               union
               {
190   0xbe        struct IOExtPar  pd_p1; /* Parallel */
190   0xbe        struct IOExtSer  pd_s1; /* Serial   */
               }
272   0x110    struct timerequest     pd_TIOR;
312   0x138    struct MsgPort         pd_IORPort;
346   0x15a    struct Task            pd_TC;
438   0x1b6    UBYTE                  pd_Stk[0x800]; /* Stack */
2486  0x9b6    UBYTE                  pd_Flags;
                                        /* OpenDevice() */
2487  0x9b7    UBYTE                  pd_pad;
2488  0x9b8    struct Preferences     pd_Preferences;
2720  0xaa0    UBYTE                  pd_WaitEnabled;
2721  0xaa1      /* here a pad byte is missing */
2722  0xaa2    }; /* defined in "devices/prtbase.h" */
```

This structure contains two additional undocumented structures. One is the DeviceData structure that makes printer device information available to the printer driver, and the other is a PrinterSegment structure that is necessary to remove the printer driver from memory using UnloadSeg().

The printer driver is loaded from the printer device with LoadSeg(). The segment is not started through CreateProc(). This is because the printer driver is actually comparable to a library that offers printer specific routines. Now back to the two structures mentioned above:

```
Offset    Structure
------    --------
          struct DeviceData
          {
0   0x00      struct Library dd_Device;
32  0x20      APTR           dd_Segment;     /* Segment list */
36  0x24      APTR           dd_ExecBase;
40  0x28      APTR           dd_CmdVectors;  /* Jump table for
                                                device commands */
44  0x2c      APTR           dd_CmdBytes;   /* Command string */
48  0x30      APTR           dd_NumCommands; /* Number of
                                                supported
                                                commands */
52  0x32  } /* defined in "devices/prtbase.h" */
```

```
Offset    Structure
------    --------
          struct PrinterSegment
          {
0  0x00      ULONG          ps_NextSegment; /* Attention! this
                                               variable is a
                                               BPTR */
4  0x04      ULONG          ps_runAlert;    /* contains moveq
                                               #0,d0:rts */
8  0x08      UWORD          ps_Version;     /* Version number */
10 0x0a      UWORD          ps_Revision;    /* Revision number */
12 0x0c      struct PrinterExtendedData ps_PED;
70 0x46   } /* defined in "devices/prtbase.h" */
```

As you can see, the PrinterSegment structure, up to the pointer ps_NextSegment, which is declared as a ULONG variable but is really a BPTR, states the header of the printer driver. Now back to the PrintData structure. Beside some internal variables, some important variables are also made available to the user:

pd_PrintBuf contains the address of the memory range needed to print a line of hardcopy. By a hardcopy line we mean the section of a hardcopy that the printer can print in a single printhead movement from left to right. An eight-pin printer produces eight lines of pixels in one pass.

pd_PWrite is the main function of the entire printer device. This sends the data through the serial or parallel interface. The SendIO() command is used for data transmission. That is why we use the double buffer printout.

The pd_PBothReady executes hardcopy printout (a hardcopy line is printed out while the next is calculated). Both print operations must be closed before execution because it may result in a system crash.

Look at the PrinterData structure, and notice that two unions exist there. They contain a device block, either for parallel or serial devices (double buffering).

An important structure in the PrinterData structure is the timer device block pd_TIOR. This device block lets you send a TR_ADDREQUEST command very easily. This is especially important after a printer reset before the hardcopy routine, because you should let at least a second elapse before you begin the hardcopy execution.

You can get all of the important Preferences structure data from the PrintData structure. You have set the data with the Preferences (e.g., line feed, etc.). Now back to the PrinterExtendedData structure:

ped_Expunge: This routine executes before the UnLoadSeg() function of the printer device. You have the option of closing the libraries opened by ped_Init, etc.

ped_Open: This routine executes after every OpenDevice() function. The difference between ped_Open and ped_Init is that ped_Init is executed after the printer driver is loaded. This is not removed from memory after it is used. For this an extra Expunge() must be called. If the printer device is already in memory the ped_Open routine is called after each OpenDevice(). This routine is given to the device block from the printer device:

```
Open (IOStdReq)
struct IOStdReq
     *IOStdReq;
{...} /* usually moveq #0,d0: rts */
```

ped_Close: This routine is called after each CloseDevice(). This routine is also given to the device block of the printer device.

7.2.1.1 Additional variables

ped_PrinterClass:

This UBYTE contains the class of the printer supported by the driver. The following values exist:

```
#define PPC_BWALPHA  0   - black and white printer (only text)
#define PPC_BWGFX    1   - black and white graphic printer
#define PPC_COLORGFX 3   - color graphic printer
```

ped_ColorClass:

This UBYTE specifies the color class of the supported printer:

```
#define PCC_BW         1 - black and white
#define PCC_YMC        2 - Yellow, Magenta, Cyan
#define PCC_YMC_BW     3 - Yellow, Magenta, Cyan or black/white
#define PCC_YMCB       4 - Yellow, Magenta, Cyan, Black
#define PCC_WB         9 - Black/White (inverted)
#define PCC_BGR       10 - Blue, Green, Red
#define PCC_BGR_WB    11 - Blue, Green, Red or Black/White
#define PCC_BGRW      12 - Blue, Green, Red and White
```

ped_MaxColumns:

This variable specifies the number of printed columns that the printer can print. If you can print 80 characters per line, for example, you must specify the value 80 here.

ped_NumCharStes:

This variable contains the number of different text types that your printer can use (e.g., pica, condensed, elite, sans, serif).

ped_NumRows:

This variable contains the number of lines that you can print per pass of the printhead. With an eight-pin dot-matrix printer this variable contains the value 8.

ped_MaxXDots:

This variable contains the number of points that fit in a print line. With 80 characters per line, and characters eight pixels wide, the value here is 8*80 = 640. If your printer supports different character widths, this value can vary.

ped_MaxYDots:

This variable contains the number of lines that fit on one page of text. This variable is only for page-oriented printers (e.g., laser printer). For dot-matrix printers with fanfold paper, the value 0 can be entered here.

ped_XDotsInch:

This variable contains the number of pixels that the printer can print horizontally per inch.

ped_YDotsInch:

This variable contains the number of pixels that the printer can print vertically per inch.

7.2.1.2 DoSpecial and command array

ped_Commands:

The command strings are contained in this array. These command strings are converted into Amiga command strings by the `CMD_WRITE` function. If, for example, the sequence "<ESC>c" appears in the data that should be sent to the printer, this is converted into the necessary sequence ("<ESC>c" resets the printer). The array has the following appearance:

```
char *CommandTable[] =
     {
       "\375\033\015P\275", /* Reset for Diablo printer */
       "...",
     };
```

The address of this array is in `ped_CommandTable`. In case one of the Amiga sequences cannot be substituted, you can give the value "\377" = 255 for this sequence in the `CommandTable` array. When this sequence emerges somewhere with the output, the Amiga knows that the translation of this sequence is possible in `DoSpecial`. When your printer does not support some of these sequences you can give the value "\377" where no processing of the sequence in `DoSpecial` occurs.

ped_DoSpecial:

This variable contains the address of the `DoSpecial` function:

```
DoSpecial (Command,OutPutBuffer,Line,LineSpace,CRLF,Params)
UWORD     *Command;
char              OutPutBuffer[];
BYTE                          *Line;
BYTE                               *LineSpace;
BYTE                                         *CRLF;
BYTE                                               Params[];
{...}
```

`Command` returns the address of the command number. `OutPutBuffer` is the memory range to which the new command string is written. To realize the paper feed and transport back, `Line` contains the address of a variable that can have the values -1, 0, 1. This variable is added to an internal variable of the printer device that contains the current print line on the paper.

When you, for example, want to paper feed one line, you must set this variable to 1 in addition to the print command for the linefeed so that the internal variable remains correct. With a paper transport back a line you must give the value -1 in `*Line`. If no paper transport should take place, the value 0 is given in `*Line`, or it is ignored.

`LineSpace` contains the address of the control character responsible for the size of the linefeed. If, for example, the sequences "<ESC>0" and "<ESC>1" establish the linefeed at 1/8 and 1/6 of an inch, `LineSpace` contains the value "0" or "1".

CRLF specifies whether a linefeed should be sent after a CR or not (*CRLF == TRUE => send linefeed). The parameters that you have given the `DoSpecial` function through the printer device command `PRD_PRTCOMMAND` are given in `Params`. When the `DoSpecial()` routine is called through the `CMD_WRITE`, no parameters are given in `Params`.

ped_Render: This routine takes over the hardcopy function. The following parameters are given:

```
Render (ct,x,y,Status)
UBYTE   ct;
UWORD      x,y;
UBYTE          Status;
{..}
```

Because this routine must assume many tasks, these variables have different interpretations. The `Render` function consists of six partial functions. It is easiest to check these six partial functions through a `SWITCH` construct:

```
switch(Status)
{
   case 0:     /* Master Initialization */
     /* x = width of the Hardcopy */
     /* y = height of the Hardcopy */
   break;
     case 1:     /* transfer point into printer buffer */
     /* ct = color of the point (Black = 0, Yellow = 1,
     /*                         Magenta = 2, Cyan = 3) */
     /* x = X Position */
     /* y = Y Position */
   break;
   case 2:     /* send print buffer to the printer */
     /* (*(pd_PWrite))(Buffer,Len); */
   break;
   case 3:     /* initialize printer buffer */
   break;
   case 4:     /* Close Down */
     /* ct = Error Code */
     /* x = Special Flags */
   break;
   case 5:     /* Pre Master initialization */
     /* x = Special Flags */
   break;
}
```

What must happen in the partial functions?

`case 0:`

This partial function reserves the necessary print buffer in which the individual pixels to be printed for the hardcopy are written. A printer reset should also be executed, followed by a one-second delay.

The size of the print buffer is determined by the number of pins of the printer. An eight-pin printer needs an 8*x byte print buffer. This print buffer must also contain the control characters for "graphic mode on" and "graphic mode off" so that the hardcopy can be sent immediately after printing the text. With double buffering this print buffer is twice as large.

case 1: This partial function transfers the pixels given by X and Y into the print buffer. You must calculate the Y coordinate of a hardcopy line so that it fits in the print buffer described above.

case 2: Here the print buffer sends graphic control characters to the printer. For this the `PWrite()` function from the `PrinterData` structure is used. This routine gives the address of the print buffer as well as the number of bytes to be output. With double buffering you must also determine the other print buffer for the transfer of the points in this section function (case 1).

case 3:

After printing the buffer it must be re-initialized. This partial function writes the control codes for "graphic on" and "graphic off" at the beginning and end of the print buffers, and deletes the previously entered pixels from the print buffer. This slows partial function 1 as little as possible, and pixels still in the print buffer are `OR`ed. Unset pixels do not change the print buffer.

case 4:

This routine waits with `(*(pd_PBothReady))()` until both print buffers (double buffering) are printed out, and then releases these and returns the error code given in `ct`.

case 5:

This partial function is the first called (even before case 0). In it you can set the `Render` internal variables to inform `case 1:` that the picture should be centered or variables for the print width is set. In x you get the special flags that are given with DUMPRPORT.

7.2.1.3 The rest of the variables

ped_TimeoutSecs:

This variable contains the number of seconds that the Amiga should wait until the printer is turned on or set to ONLINE. When this time elapses, the printer trouble requester appears. If you click on CANCEL in this requester the `PDERR_CANCEL` error is sent.

ped_8BitChars:

This pointer contains the address of a 256-byte array that contains the ASCII codes to be printed when using the extended fonts.

7.2.2 Tips for programming a printer driver

These tips apply if you use the Lattice C compiler for developing your printer drivers. All of the parameters given in Lattice use the long word format. That means that a four-byte UBYTE memory is needed to access the bottom byte of this long word. The Lattice compiler reserves registers a2-a5 and registers d2-d6 from the call. Should you develop a driver, you should be aware of this and adapt any byte access to Lattice's quirks.

Although many of the functions described above are of type `VOID` (they have no return value) you must return a zero in d0 to the routine, if it executed without an error. If an error occurs, you should return the printer device error. Usually the routines `Open` and `Close` are unnecessary. That is why these should have the following appearance when they are not used:

```
{
   return(0l);
}
<<or>>
   moveq #0,d0
   rts
```

7.3 Fonts and text output

You probably have toyed with the thought of creating your own fonts. Creating a font is as easy as booting a font editor, but you should know what lies behind the Amiga fonts.

A font is available in different point sizes. Take the Opal font, for example. This font comes in 12 and 9 sizes. If you look in the `Fonts` directory of the SYS: diskette using the CLI `dir` command, you'll find the following entry:

```
Opal (dir)
Opal.font
```

The file `Opal.font` is the font header. It contains the Opal font data for the 12-pixel and 9-pixel sizes. The `Opal` directory contains the files `Opal.12` and `Opal.9`, which contain the character definitions.

7.3.1 The font header

The font header has the following structure:

```
FontContentsHeader
FontContents (for example for font size 12)
FontContents (for example for font size 9)
```

The `FontContentsHeader` structure informs the Amiga that it handles this file as a font header, and how many fonts it manages. This structure looks like this:

```
Offset   Structure
------   --------
         struct FontHeader
         {
0 0x00        UWORD fch_FileID;
2 0x00        UWORD fch_NumEntries;
4 0x00   } /* defined in "libraries/diskfont.h" */
```

`fch_FileID` contains the value 0x0f00 and signals the Amiga that it handles this file as a font header.

`fch_NumEntries` contains the total number of fonts the Amiga has available. It only lists as many fonts that exist under a certain name, e.g., Opal. Because the Opal font is present in two point sizes (9 and

12), this entry has the value 2 for the Opal font. The `FontContents` structure supplies the exact information about each font:

```
Offset    Structure
------    --------
          struct FontContents
          {
0   0x000    char  fc_FileName[256];
256 0x100    UWORD fc_YSize;
258 0x102    UBYTE fc_Style;
259 0x103    UBYTE fc_Flags; 260 0x104 } /* defined in
                                           libraries/diskfont.h */
```

`fc_FileName` contains the filenames for the font types (e.g., Opal/12). Through these filenames the Amiga can load and use the fonts. A `FontContents` structure must be stored for each font type. The rest of the bytes are filled with zeros.

`fc_YSize` contains the size of the font. This variable can be found very quickly when opening the font if the required font is present.

`fc_Style` contains the text style of the font. Normally the value zero is here, which says that the font is saved in normal style. You can, for example, define the individual characters so that they appear as italics. Then you should set the `FSF_ITALIC` flag (4) so that the Amiga knows that an italic font in the size `YSize` is found in the `fc_FileName` file. If you want to open a font that is eight lines high and should be italic, the Amiga can recognize the header to see if such a font exists.

If this is not the case, the Amiga tries to load the font that comes the closest to the given size. If this font only contains normal characters, the font can be italicized through software if needed. The following text types are recognized by the Amiga, or can be created with the software:

FSF_ITALIC (4):

The top half of the character is pushed a bit to the right.

FSF_BOLD (2):

The characters are displayed normally and pushed to the right with `tf_BoldSmear` pixels.

FSF__UNDERLINED (1):

Here a line is drawn in the baseline.

FC_FLAGS:

Contains the flags that inform the Amiga in the font's status.

FPF_ROMFONT (1):

The font is stored in ROM. This flag does not apply to us because we are only concerned with disk supported fonts. The single font in ROM is the Topaz font.

FPF_DISKFONT (2):

The font is stored on disk. This flag must be set in the `FontContents` structure of the `flags` variable.

FPF_PROPROTIONAL (32):

The font supports proportional text (variable character widths).

FPF_REMOVED (128):

This flag announces that the font of the system is not ready. Because that is not encountered for the disk fonts that are not opened, this flag must also be set.

The following is a short C program that initializes the necessary structures for a font header and saves this to disk.

```
/****************************************************************
*                          Font.c                               *
*                        August 1988                            *
*                     (c) Bruno Jennrich                        *
*                                                               *
* Function: Create Font-Header                                  *
****************************************************************/
/****************************************************************
* Compile-Info:                                                 *
*                                                               *
* cc Font.c                                                     *
* ln Font.o -lc                                                 *
****************************************************************/

#include "exec/types.h"
#include "exec/memory.h"
#include "libraries/dos.h"
#include "libraries/diskfont.h"
#include "graphics/text.h"

#define NUMFONTS 1

VOID *Open();

#define FONCON_LEN (ULONG) sizeof (struct FontContents)
#define FONHED_LEN (ULONG) sizeof (struct FontContentsHeader)

BYTE  *FontHeaderName = "OwnFont.font";
```

```
struct FontContentsHeader
       FontContentsHeader[NUMFONTS] = {0x0f00,
                                       /* File is Font-Header */
                                       0x0001  /* Single Font */
                                      };
struct FontContents
       FontContents[ANZFONTS] = {
                                {"OwnFont/9",
                                 0x0009,
                                 0x00,
                                 FPF_REMOVED|FPF_DISKFONT
                                }
                               };

/****************************************************************
*                            main                               *
*                                                               *
****************************************************************/

main()
{
   UWORD *FileHandle = 0l;
   UWORD i,j;


   FileHandle = Open (FontHeaderName, (ULONG)MODE_NEWFILE);
   if (FileHandle == 0L)
   {
      printf ("No File !!!\n");
      exit (0);
   }

   for (j=0;j<ANZFONTS;j++)
   {
      if (j==0)
         Write (FileHandle,&FontContentsHeader[0],FONHED_LEN);

      i = strlen (FontContents[j].fc_FileName);
      for (   ;i<256;i++)       /* fill with  null bytes */
         FontContents[j].fc_FileName[i] = 0x00;

      Write (FileHandle,&FontContents[j],FONCON_LEN);
   }

   Close (FileHandle);
}
```

Note: You must store your own `FontContents` structure for each size of the font (e.g., Opal-12, Opal-9). The font header created from the above program must be copied into the `Font` directory, and then the header determines the appearance of each character.

7.3.2 The actual character data

The actual character data is saved in files. When you look at the Opal subdirectory for instance, there you find files named 12 and 9. These files contain the actual data. Let's create a file like this.

The Amiga handles these files as normal programs. You can call a font file like a normal program—only nothing happens. The first four bytes of these programs contain the commands:

```
moveq #0, d0
rts
```

You have stored the fonts as program files to be able to load and remove these simply with `LoadSeg()` and `UnLoadSeg()`. This makes the two above commands necessary, so that a system interrupt does not happen when starting. A `FontData` file has the following appearance:

```
moveq #0, d0
rts
DiskFontHeader
FontData
```

Let's examine the disk font header. This structure has the following appearance:

```
Offset    Structure
------    ---------
          struct DiskFontHeader
          {
0  0x00      struct Node     dfh_Node;
14 0x0e      UWORD           dfh_FileID;
16 0x10      UWORD           dfh_Revision;
18 0x12      LONG            dfh_Segment;
22 0x16      char            dfh_Name[32];
52 0x34      struct TextFont dfh_TF;
          } /* defined in "libraries/diskfont.h" */
```

`dfh_Node` adds the font to the system font list after it is loaded. After that the Amiga no longer needs to access the disk, but the user can still access the font.

`dfh_FileID` contains a label that informs the Amiga to handle the file as a font file. This label has the value 0x0f80.

`dfh_Revision` can be used to assign your own private font version number. The value is usually here.

`dfh_Segment` contains the address of the segment list that is returned after `LoadSeg()`. It is necessary to be able to remove the font from the system after it is used with `UnLoadSeg()`. This value must be set to zero by you. The Amiga does the rest.

`dfh_Name` contains the name of the fonts so that the `OpenFont()` command can recognize that the font to be opened is found in the memory.

`dfh_TF` is a `TextFont` structure which returns a pointer after `OpenFont()` or `OpenDiskFont()`. The font can then use this pointer.

This `TextFont` structure (`dfh_TF`) must be initialized by the user. It looks like this:

```
Offset    Structure
------    --------
          struct TextFont
          {
0  0x00      struct Message tf_Message; /* important for
                                           UnLoadSeg */
20 0x14      UWORD          tf_YSize;
22 0x16      UBYTE          tf_Style;
23 0x17      UBYTE          tf_Flags;
24 0x18      UWORD          tf_XSize;
26 0x1a      UWORD          tf_Baseline;
28 0x1c      UWORD          tf_BoldSmear;
30 0x1e      UWORD          tf_Accessors;
32 0x20      UBYTE          tf_LoChar;
33 0x21      UBYTE          tf_HiChar;
34 0x22      UWORD          tf_Modulo;
36 0x24      APTR           tf_CharLoc;
40 0x28      APTR           tf_CharSpace;
44 0x2c      APTR           tf_CharKern;
48 0x30   } /* defined in "graphics/text.h" */
```

`tf_YSize`, `tf_Style` and `tf_Flags` contain the same values as the variables in the `FontContents` structure of the same names. `tf_XSize` contains the width of a character in pixels. When you use proportional fonts, this value changes from character to character. For example, the "!" in a proportional font may be only three pixels wide, while the "A" in the same font may be nine pixels wide. You should specify the maximum width of a character in `tf_XSize` when using proportional fonts. Many word processors use this value to write the characters in a matrix, which cannot be done with proportional fonts.

`tf_Baseline` contains the line on which the character is positioned. This baseline is given in lines from the top line of the character. When you determine the position of a character to be given with `Move (RastPort, 0 ,10)`, for example, the baseline of the character is at line 10 of the RastPort. The baseline must also act as an underline for software reasons.

`tf_BoldSmear` gives the smear factor for bold type.

`tf_Accessors` contains the number of the user currently accessing the font. If this variable contains a value of zero, the memory allocated for the font can be freed.

`tf_LoChar` contains the ASCII code of the first defined character.

`tf_HiChar` contains the ASCII code of the last character generated.

`tf_Module` supplies the width of the character array.

`tf_CharSpace` points to the UWORD array that determines the width of each character.

`tf_CharKern` points to a WORD array that determines the position at which the character data should be displayed.

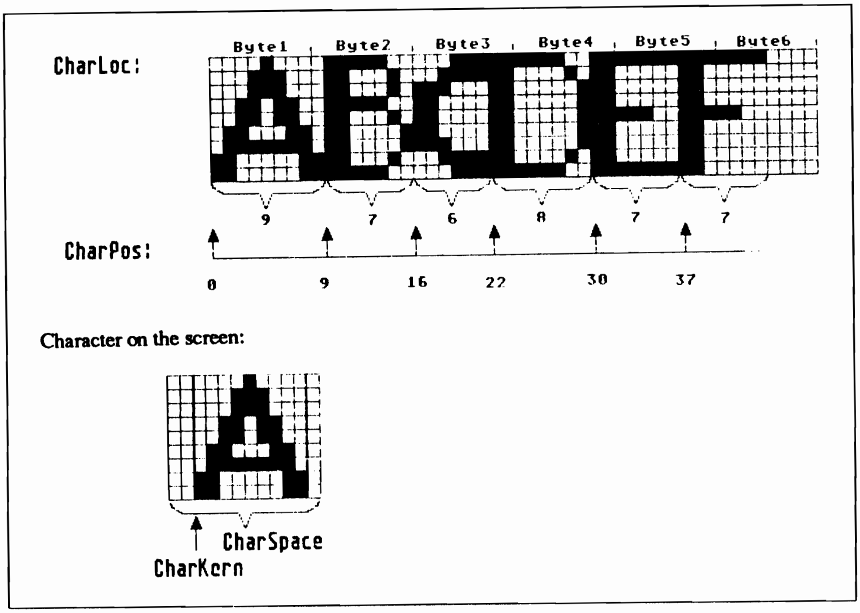
Say you want to create a font that consists of capital letters only. The `tf_LoChar` and `tf_HiChar` variables would then contain the values 65 decimal (A) and 91 (Z) respectively. Your new characters are then saved in an array that looks like this:

```
Byte  1, 2, 3, 4, 5, 6, 7, 8
Byte  9, 10, 11, 12, 13, 14, 15, 16
...
Byte 56, 57, 58, 59, 60, 61, 62, 63
```

Bytes 1-8 contain the top lines of all of the characters. Bytes 9-16 contain the second lines of all the characters, and so on. You see, the character array is organized like a bit-map. The Blitter chip must know the size of the array so that the Blitter can correctly read the data for a character from this array. In this case the width would be eight bytes. You must then give the value 8 for `tf_Module`.

Because the individual characters in the character array are saved without any spaces in between (to save memory), the `CharSpace` array allows display of each character. This array contains the width of each character, as well as the width of the character container into which the characters from the `CharLoc` array are copied.

The `CharKern` array contains the column to which the characters in this container are copied. The following diagram shows this connection:



Now you have the elements needed to create your own font. The following assembly language program contains a font that redefines the capital letters A through F. Remember that you must always define one character more than needed, that always appears when an ASCII code not included in the font is encountered (see `LoChar` and `HiChar` above). Undefined characters on the keyboard are presented as rectangles.

```
*************************************************************
*                          9.asm                            *
*                       August 1988                         *
*                   (c) Bruno Jennrich                      *
*                                                           *
*************************************************************
*************************************************************
* Info:                                                     *
*                                                           *
* as 9                                                      *
* ln 9.o                                                    *
*************************************************************

   moveq #0,d0 ; f|r versehentlichen Start
   rts

   ; Node-Structure for oner binding in System font list
   dc.l 0      ;ln_Succ
   dc.l 0      ;ln_Pred
   dc.b 12     ;ln_Type (NT_FONT)
   dc.b 0      ;ln_Pri
   dc.l Name   ;ln_Name
   dc.w $0f80  ;DiskFontHeader-ID (DFH_ID)
   dc.w 1      ;Revision
   dc.l 0      ;Segment
Name:
   dc.b
"OwnFont/9",0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

FONT:
   dc.l 0          ;ln_Succ              ;Message-Structure
   dc.l 0          ;ln_Pred
   dc.b 12         ;ln_Type (NT_FONT)
   dc.b 0          ;ln_Pri
   dc.l Name       ;ln_Name
   dc.l 0          ;mn_ReplyPort
   dc.w Ende-FONT  ;Length
   dc.w 9          ;YSize
   dc.b 0          ;tf_Style (before defined?)
   dc.b 128+32+2   ;tf_Flags
(FPF_REMOVED+FPF_PROPORTIONAL+FPF_ROMFONT)
   dc.w 11         ;tf_XSize
   dc.w 7          ;tf_Baseline
   dc.w 1          ;tf_BoldSmear
   dc.w 0          ;tf_Accessors
   dc.b 65         ;tf_LoChar 'A'
   dc.b 70         ;tf_HiChar 'F'
   dc.l CharData   ;tf_CharData
   dc.w 8          ;tf_Modulo
   dc.l CharLoc    ;tf_CharLoc
   dc.l CharSpace  ;tf_CharSpace
   dc.l CharKern   ;tf_CharKern

CharData:
      ;0123456789012345  6789012345678901  2345678901234567  8901234567890123
   dc.w %0000110000111111,%0000111111111111,%1100111111111111,%1111111111111111
   dc.w %0001111000110001,%1001100000110000,%0110110000000110,%0000001000000001
   dc.w %0001111000110001,%1011000000110000,%0011110000000110,%0000001000000001
   dc.w %0011001100111111,%0011000000110000,%0011110000000110,%0000001000000001
   dc.w %0011001100110001,%1011000000110000,%0011111111000111,%1111001000000001
   dc.w %0111111110110000,%1111000000110000,%0011110000000110,%0000001000000001
   dc.w %0110000110110000,%1111000000110000,%0011110000000110,%0000001000000001
   dc.w %1100000011110001,%1001100000110000,%0110110000000110,%0000001000000001
   dc.w %1100000011111111,%0000111111111111,%1100111111111110,%0000001111111111
   ;     ^          ^          ^        ^             ^         ^          ^
   ;     A          B          C        D             E         F          undefined
```

```
CharLoc:
   dc.w 0,10          ;A
   dc.w 10,8          ;B
   dc.w 18,8          ;C
   dc.w 26,10         ;D
   dc.w 36,9          ;E
   dc.w 45,9          ;F
   dc.w 54,10         ;undefined

CharSpace:
   dc.w 11            ;A
   dc.w 9             ;B
   dc.w 9             ;C
   dc.w 11            ;D
   dc.w 10            ;E
   dc.w 10            ;F
   dc.w 11            ;undefined

CharKern:
   dc.w 1
   dc.w 1
   dc.w 1
   dc.w 1
   dc.w 1
   dc.w 1
   dc.w 1

Ende:
   end
```

Because font files must be stored as program files, the following steps must be taken:

```
as 9.asm
ln 9.o
```

The linking is necessary to store the file as a program file (including all of the hunks) so that it can be loaded using `LoadSeg()`. Then we must copy the finished program file to a subdirectory of the `Font` directory (you must create a subdirectory within the `Font` directory using the CLI `makedir` command). Then you can use the font.

The C program printed in this chapter (`Font.c`) creates a header file for the font `OwnFont`. To use this font you must place the subdirectory `OwnFont` in the Fonts directory and copy the assembled version of the font created above (9). The font header also must be copied to the `Font` directory.

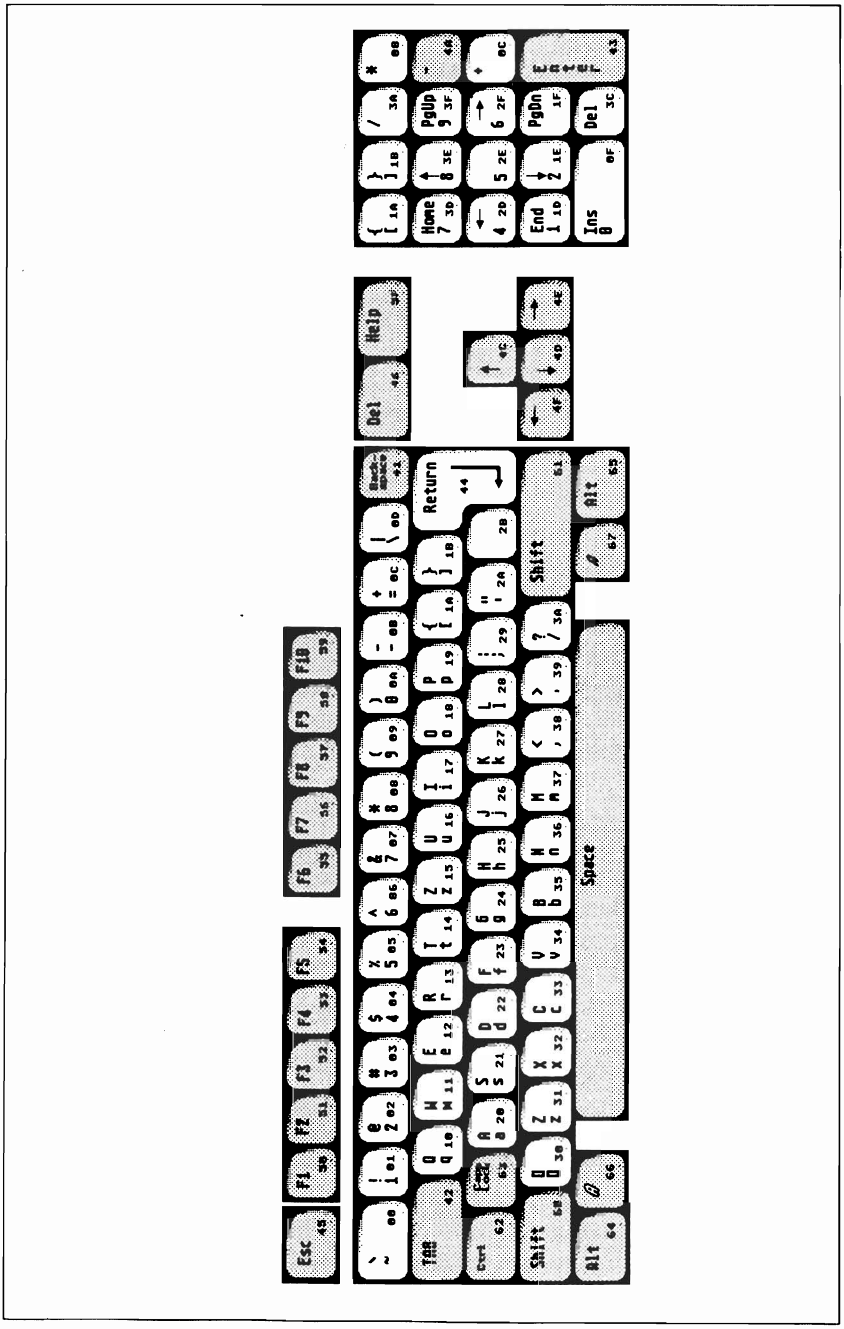
7.4 Keymaps

It is important that the keyboard of a computer be adaptable to fit other languages, because some countries use different keyboard arrangements. Some of the keys must be transposed (e.g., Y becomes Z and Z becomes Y). This alone is not enough. The Amiga keyboard, which includes a small microprocessor, tells the Amiga that a certain key was pressed, rather than a certain character. The Amiga does not know which character is represented by this key. The number of the key sent to the Amiga is called the RAWKEY number.

The following diagram of the International Amiga 500 and Amiga 2000 keyboard, includes the RAWKEY numbers in hexadecimal notation. The Amiga International keyboard has two extra keys, one by the <Return> key and one by the <Left Shift> key. The operating system converts these RAWKEY numbers into key characters. The system uses a table called a keymap as a reference, into which the character representing each RAWKEY number is entered. The Amiga includes keymaps for a number of different countries. A keymap is usually loaded during the boot operation, or by the CLI `SetMap` command. If no keymap is found the Amiga defaults to an American (USA) keymap.

In addition to regular characters, the keymap indicates whether a key reacts to the <Caps Lock> key. An active <Caps Lock> key usually has the same effect as continually pressing a <Shift> key. This means that you get upper case letters instead of lower case letters when you press the letter keys. If you press number keys you may also get the corresponding special characters (!,",#,$,...). Since most people want access to upper case letters and numbers, but not the shifted numbers, you can exclude any keys from the <Caps Lock> function, including number and cursor keys.

The keymap also assigns key repeating for individual keys. For example, if you press a cursor key, the key continues to repeat until you release the key. This lets you move the cursor easily in a word processor. The <Return> key has no repeat function since it is used mainly for input, and shouldn't have a repeat function assigned to it.


Esc
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
Tab
Ctrl
Shift
Alt
Space
Return
Back-space
Del
Help
Home
PgUp
End
PgDn
Ins
Enter

All of these parameters are present in the keymap for each key. A keymap has the following structure:

```
struct KeyMap
{
0x00  0  UBYTE *km_LoKeyMapTypes;
0x04  4  ULONG *km_LoKeyMap;
0x08  8  UBYTE *km_LoCapsable;
0x0C 12  UBYTE *km_LoRepeatable;
0x10 16  UBYTE *km_HiKeyMapTypes;
0x14 20  ULONG *km_HiKeyMap;
0x18 24  UBYTE *km_HiCapsable;
0x1C 28  UBYTE *km_HiRepeatable;
0x20 32
}
```

The parameters of the keys containing a RAWKEY number from 0 to 0x3f are in `LowKeyMap`, while the rest belong to the `HighKeyMap`.

The keymap points to the table which contains an entry for each key. The `KeyMapTypes` pointer includes a byte for each key from which is determined which character the key delivers in conjunction with <Shift>, <Alt> and <Ctrl>. These characters are stored in the second table and occupy one long word per key. One bit is reserved for each key in the third table. This bit is set to 1 if the key should react to <Caps Lock>. The fourth and final table includes a bit to indicate if the key has a repeat function.

The bit assignment of the last two tables to the RAWKEY numbers is intuitive: Bit 0 of byte 0 of the table belongs to the RAWKEY number 0 (`km_Lo`; with `km_Hi` you must add 0x40 to the RAWKEY number), bit 1 of byte 0 belongs to RAWKEY number 1, bit 0 of byte 1 to RAWKEY number 8 and so on. The RAWKEY number divided by eight results in the byte number, and the bit number is the remainder (modulo eight).

The `KeyMapTypes` includes the following, which tell if the corresponding key reacts to a qualifier (<Shift>, <Alt> or <Ctrl>):

```
KC_NOQUAL       0  Reacts to no other key
KC_VANILLA      7  Reacts to Shift, Alt and Ctrl
KCF_SHIFT    0x01  Reacts to Shift
KCF_ALT      0x02  Reacts to Alt
KCF_CONTROL  0x04  Reacts Ctrl
KCF_DOWNUP   0x08  Reacts first after key release
KCF_DEAD     0x20  Special key for special characters
KCF_STRING   0x40  Displays string instead of character
KCF_NOP      0x80  No reaction
```

Depending on which flags are set, the corresponding character is in the long word of the second table, `km_LoKeyMap` or `km_HiKeyMap`.

Qualifier	0. Byte	1. Byte	2. Byte	3. Byte
KC_NOQUAL	-	-	-	alone
KCF_SHIFT	-	-	Shift	alone
KCF_ALT	-	-	Alt	alone
KCF_CONTROL	-	-	Ctrl	alone
KCF_ALT+KCF_SHIFT	Shift+Alt	Alt	Shift	alone
KCF_CONTROL+KCF_ALT	Ctrl+Alt	Alt	Ctrl	alone
KCF_CONTROL+KCF_SHIFT	Ctrl+Shift	Ctrl	Shift	alone
KC_VANILLA	Shift+Alt	Alt	Shift	alone

KC_VANILLA, which is only a combination KFC_SHIFT, KCF_ALT and KCF_CONTROL, adds a special effect to the letter keys and a few other keys. When the <Ctrl> key is pressed, bits 5 and 6 are cleared from the character code.

If the KCF_STRING flag is set, the long word is handled as a pointer to a string descriptor (more on this later) instead of four individual bytes. The following are a few examples from a German keymap:

```
RAWKEY-Nummer: 0x0E
KeyMapType:    0x80
```

This means that this RAWKEY number is not allocated because there is no key that sends this number.

```
RAWKEY Number: 0x30
KeyMapType:    0x01         Reacts only to the shift key.
KeyMap:        0,0,'>','<'
Capsable:      0
Repeatable:    1
```

This key usually sends the < character. If the <Shift> key is pressed in addition, the result of the key is the > character. The <Caps Lock> key does not react with this key, and for that reason it has a repeat function.

```
RAWKEY-Nummer: 0x45
KeyMapType:    0x02        Reacts to the Alt key.
KeyMap         0,0,0x9B,0x1B
Capsable:      0
Repeatable:    0
```

This key (ESCAPE key) usually delivers 0x1b (27). When <Alt> is pressed at the same time, you get 0x9B. This key does not react to the <Caps Lock> key and does not have a repeat function.

```
RAWKEY-Nummer: 0x01
KeyMapType:    0x03        Reacts to the Shift and Alt keys.
KeyMap:        '!',0xB9,'!','1'
Capsable:      0
Repeatable:    1
```

This key without a qualifier results in a 1. When you press <Shift> with it, you get the !, with the <Alt> you get the character with the code 0xB9 and when you press the <Shift> and <Alt> key at the same time, you get the ! again. The key reacts to the <Caps Lock> key and has a repeat function for this.

```
RAWKEY-Nummer: 0x10
KeyMapType:    0x07        Reacts to Shift, Alt, Ctrl.
KeyMap:        0xC5,0xE5,'Q','q'
Capsable:      1
Repeatable:    1
```

This key is a normal letter key. Without a qualifier you get a small letter and with a qualifier you get a capital letter. When you press <Alt>, you get a special character, 0xC5 and with <Shift> (or with the activated <Caps Lock> key) 0xE5. When you press <Ctrl> you get the code of the character without a qualifier, but with the 5th and 6th bits cleared, q = 0x71 => 0x71 & 0x9F = 0x11 = ^Q. You always get this code when you have pressed <Ctrl> at the same time, regardless of whether you have also pressed <Shift> or <Alt>. This key reacts to <Caps Lock> and has a repeat function.

The following keys supply strings, so that we should look at the construction of a string descriptor. Just as each key can send different characters, this can also send different strings. For each combination with one allowable qualifier there is an entry in the string descriptor made up of two bytes, just like normal keys. The first byte gives the length of the string and the second gives the distance of the string from the beginning of the string descriptor. Unlike normal keys, the order here is exchanged, which means the string for the key without the qualifier is described first, then for the key with <Shift>, then with <Alt>, <Shift> plus <Alt>, etc.

```
RAWKEY-Nummer: 0x5F
KeyMapType:    0x40        String.
KeyMap:        Pointer to  string descriptor
Stringdescr.:  0x03,0x02
               0x9B,0x3F,0x7E
Capsable:      0
Repeatable:    0
```

This key (<Help>) supplies a string of length three: 0x9B 0x3F 0x7E.

```
RAWKEY-Nummer: 0x42
KeyMapType:    0x41        String, reacts to Shift.
KeyMap:        Pointer to String descriptor.
Stringdescr.:  0x01,0x04   without Shift (length = 1)
               0x02,0x05   with Shift (Length = 2)
               0x09        String without Shift (distance = 4)
               0x9B,0x5A   String with Shift (distance = 5)
Capsable:      0
Repeatable:    1
```

This key, the <Tab> key, usually sends a character with code 9. When pressed with the <Shift> key, the <Tab> key sends the string 0x9B 0x5A.

Complicated strings are not offered by the keymap. For these we must access the dead keys. The following example does not work with the USA keymaps, use `SETMAP d` to enable the German keymap. These are especially interesting when used in conjunction with the equals key (RAWKEY number 0x0C, the apostrophe key in the German keymap). If you press this key, nothing happens. If you press a vowel key (<a>, <e>, <i>, <o>, <u>), <Space>, <y> or <n>, the system displays the corresponding character with an accent (try this from the CLI).

All of these keys (<'>, vowels, <Space>, <y> and <n>) are marked as dead keys in the `KeyMapTypes`. When a key is a dead key, the keymap long word contains a pointer that points to a dead key descriptor. This consists of a row of 2 byte entries from which the first byte gives how the second is handled:

0 The second byte is the character code that sends this key with the corresponding qualifiers.

1 This key (`DFP_MOD`) is modified through a dead key. The second byte is an offset from the beginning of the dead key descriptor field to a field of 18 bytes (this number varies from keymap to keymap), that the codes the characters contain. These keys can send the codes depending on which dead keys were pressed.

8 This key (`DPF_DEAD`) is the actual dead key. The second byte contains an offset that designates which character is used from the 18-byte field. This offset is noted in the operating system until another key is pressed. When this key is handled as a dead key, this offset is ignored. The offset itself is actually divided into two nibbles. When the high nibble (bits 7 through 4) is unequal to zero, the value of this nibble is multiplied by the low byte, and both are added together with the previous offset. When the previous offset of the high nibble was unequal to zero, only the low nibble is added. If no previous offset exists, zero is added and only the product is used.

Here are a few examples that hopefully will make it clearer:

```
RAWKEY Number: 0x23
KeyMapType:    0x27         Dead and Vanilla
KeyMap:        pointer to dead key descriptor
Dead-Key-Des.: 0,'f'        without qualifier
               0,'F'        with Shift
               8,0x61       Alt (dead key)
               8,0x61       Alt+Shift (dead key)
               0,0x06       Ctrl
               0,0x06       Ctrl+Shift
               0,0x86       Ctrl+Alt
```

```
                0,0x86       Ctrl+Alt+Shift
Capsable:       1
Repeatable:     1
```

This key supplies either a normal character or an offset as a dead key when you press it together with the <Alt> key. The same goes for the next key:

```
RAWKEY Number: 0x24
KeyMapType:     0x27         Dead and Vanilla
KeyMap:         Pointer to dead key descriptor
Dead-Key-Des.: 0,'g'        without qualifier
                0,'G'        with Shift
                8,0x62       Alt (dead key)
                8,0x62       Alt+Shift (dead key)
                0,0x07       Ctrl
                0,0x07       Ctrl+Shift
                0,0x87       Ctrl+Alt
                0,0x87       Ctrl+Alt+Shift
Capsable:       1
Repeatable:     1
```

This key reacts the same way as mentioned before, but it has a different offset than the dead key (and also sends a different character). Let's take a third dead key:

```
RAWKEY Number: 0x25
KeyMapType:     0x27         Dead and Vanilla
KeyMap:         Pointer to dead key descriptor
Dead-Key-Des.: 0,'h'        without qualifier
                0,'H'        with Shift
                8,0x03       Alt (dead key)
                8,0x03       Alt+Shift (dead key)
                0,0x08       Ctrl
                0,0x08       Ctrl+Shift
                0,0x88       Ctrl+Alt
                0,0x88       Ctrl+Alt+Shift
Capsable:       1
Repeatable:     1
```

This key has a dead key offset where the high byte nibble is zero. Now here is a key modified through the dead keys:

```
RAWKEY Number: 0x20
KeyMapType:     0x27         Dead and vanilla
KeyMap:         Pointer to dead key descriptor
Dead-Key-Des.:
                1,0x10       without qualifier (dead modified)
                1,0x22       Shift (dead modified)
                0,0xE6       Alt
                0,0xC6       Alt+Shift
                0,0x01       Ctrl
                0,0x01       Ctrl+Shift
                0,0x81       Ctrl+Alt
                0,0x81       Ctrl+Alt+Shift
```

```
                 Field for key without qualifier:
                 'a', 0xE0,0xE1,0xE2,0xE3,0xE4
                 0xE0,0xE0,0xE2,0xE0,0xE0,0xE0
                 0xE1,0xE2,0xE1,0xE1,0xE1,0xE1
                 Field for key with Shift:
                 'A', 0xC0,0xC1,0xC2,0xC3,0xC4
                 0xC0,0xC0,0xC2,0xC0,0xC0,0xC0
                 0xC1,0xC2,0xC1,0xC1,0xC1,0xC1
Capsable:        1
Repeatable:      1
```

Now let's look at a few options offered by different field entries, and which keys you must press:

```
A          No offset => 'a' (character in position zero)
Alt-F      Offset = 0x61
A          Low nibble of offset, you get 0xE0
Alt-G      Offset = 0x62
Shift-A    Low nibble of offset = 2, result => 0xC1
Alt-H      Offset = 3
Shift-A    Offset = 3, result => 0xC2
Alt-F      Offset = 0x61
Alt-G      Offset = Offset & 0x0F + (6 * 2) = 13
A          Character in position 13 0xE2
Alt-G      Offset = 0x62
Alt-F      Offset = Offset & 0x0F + (6 * 1) = 8
A          Character in position 8 0xE2
Alt-G      Offset = 0x62
Alt-H      Offset = 3
A          Character in position 3 0xE2
Alt-H      Offset = 3
Alt-F      Offset = Offset & 0x0F + (6 * 1) = 9
A          Character in position 9 0xE0
Alt-H      Offset = 3
Alt-G      Offset = Offset & 0x0F + (6 * 2) = 15
Shift-A    Character in position 15 0xC1
```

Now let's show you an example of a keymap in source form. The following assembler source code is a disassembly of a German keymap used on the Amiga:

```
;
; German Keymap
; (disassembled)
;
;Node:
 dc.l 0,0
 dc.w 0
 dc.l name
;Keymap:
 dc.l lotypes
 dc.l lokeymap
 dc.l locaps
 dc.l lorepeat
 dc.l hitypes
 dc.l hikeymap
 dc.l hicaps
 dc.l hirepeat
locaps:
 dc.b 0,0,$FF,7,$FF,7,$FE,0
```

```
hicaps:
 dc.b 0,0,0,0,0,0,0,0
lorepeat:
 dc.b $FF,$BF,$FF,$EF,$FF,$EF,$FF,$F7
hirepeat:
 dc.b $47,$F4,$FF,$7F,0,0,0,0
lotypes:
 dc.b $07,$03,$03,$03,$03,$03,$03,$03
 dc.b $03,$03,$03,$03,$03,$07,$80,$00
 dc.b $07,$07,$27,$07,$07,$07,$27,$27
 dc.b $27,$07,$07,$03,$80,$00,$00,$00
 dc.b $27,$07,$07,$27,$27,$27,$27,$27
 dc.b $07,$07,$07,$05,$80,$00,$00,$00
 dc.b $01,$27,$07,$07,$07,$07,$27,$07
 dc.b $03,$03,$07,$80,$00,$00,$00,$00
hitypes:
 dc.b $22,$00,$41,$00,$04,$02,$00,$80
 dc.b $80,$80,$00,$80,$41,$41,$41,$41
 dc.b $41,$41,$41,$41,$41,$41,$41,$41
 dc.b $41,$41,$05,$05,$00,$00,$00,$40
 dc.b $80,$80,$80,$80,$80,$80,$80,$80
 dc.b $80,$80,$80,$80,$80,$80,$80,$80
 dc.b $80,$80,$80,$80,$80,$80,$80,$80
lokeymap:
 dc.b '~','`','~','`'
 dc.b '!',$B9,'!','1'
 dc.b $B2,'@','"','2'
 dc.b '#',$B3,$A7,'3'
 dc.b $A2,$B0,'$','4'
 dc.b '%',$BC,'%','5'
 dc.b '^',$BD,'&','6'
 dc.b '&',$BE,'/','7'
 dc.b '*',$B7,'(','8'
 dc.b '(',$AB,')','9'
 dc.b ')',$BB,'=','0'
 dc.b '_','-','?',$DF
 dc.l deadapostroph
 dc.b '|','\','|','\'
 dc.b $00,$00,$00,$00
 dc.b $00,$00,$00,'0'
 dc.b $C5,$E5,'Q','q'
 dc.b $B0,$B0,'W','w'
 dc.l deade
 dc.b $AE,$AE,'R','r'
 dc.b $DE,$FE,'T','t'
 dc.b $A5,$A4,'Z','z'
 dc.l deadu
 dc.l deadi
 dc.l deado
 dc.b $B6,$B6,'P','p'
 dc.b '{','[',$DC,$FC
 dc.b '}',']','*','+'
 dc.b $00,$00,$00,$00
 dc.b $00,$00,$00,'1'
 dc.b $00,$00,$00,'2'
 dc.b $00,$00,$00,'3'
 dc.l deada
 dc.b $A7,$DF,'S','s'
 dc.b $D0,$F0,'D','d'
 dc.l deadf
 dc.l deadg
 dc.l deadh
```

```
 dc.l deadj
 dc.l deadk
 dc.b $A3,$A3,'L','l'
 dc.b ':',';',$D6,$F6
 dc.b '"',$27,$C4,$E4
 dc.b '^','#','^','#'
 dc.b $00,$00,$00,$00
 dc.b $00,$00,$00,'4'
 dc.b $00,$00,$00,'5'
 dc.b $00,$00,$00,'6'
 dc.b $00,$00,'>','<'
 dc.l deadz
 dc.b $F7,$D7,'X','x'
 dc.b $C7,$E7,'C','c'
 dc.b $AA,$AA,'V','v'
 dc.b $BA,$BA,'B','b'
 dc.l deadn
 dc.b $BF,$B8,'M','m'
 dc.b '<',',',';',','
 dc.b '>',$2E,':',$2E
 dc.b '?','/','_','-'
 dc.b $00,$00,$00,$00
 dc.b $00,$00,$00,$2E
 dc.b $00,$00,$00,'7'
 dc.b $00,$00,$00,'8'
 dc.b $00,$00,$00,'9'
hikeymap:
 dc.l deadspace
 dc.b $00,$00,$00,$08
 dc.l strtab
 dc.b $00,$00,$00,$0D
 dc.b $00,$00,$0A,$0D
 dc.b $00,$00,$9B,$1B
 dc.b $00,$00,$00,$7F
 dc.l $00000000
 dc.l $00000000
 dc.l $00000000
 dc.b $00,$00,$00,'-'
 dc.l $00000000
 dc.l strcdown
 dc.l strcup
 dc.l strcright
 dc.l strcleft
 dc.l strf1
 dc.l strf2
 dc.l strf3
 dc.l strf4
 dc.l strf5
 dc.l strf6
 dc.l strf7
 dc.l strf8
 dc.l strf9
 dc.l strf10
 dc.b $00,$00,'{','['
 dc.b $00,$00,'}',']'
 dc.b $00,$00,$00,'/'
 dc.b $00,$00,$00,'*'
 dc.b $00,$00,$00,'+'
 dc.l strhelp
 dcb.l 24,0
;Strings and Dead-Keys
deadapostroph:
```

```
 dc.b $08,$62,$08,$61,$00,'=',$00,'+'
deadf:
 dc.b $00,'f',$00,'F',$08,$61,$08,$61
 dc.b $00,$06,$00,$06,$00,$86,$00,$86
deadg:
 dc.b $00,'g',$00,'G',$08,$62,$08,$62
 dc.b $00,$07,$00,$07,$00,$87,$00,$87
deadh:
 dc.b $00,'h',$00,'H',$08,$03,$08,$03
 dc.b $00,$08,$00,$08,$00,$88,$00,$88
deadj:
 dc.b $00,'j',$00,'J',$08,$04,$08,$04
 dc.b $00,$0A,$00,$0A,$00,$8A,$00,$8A
deadk:
 dc.b $00,'k',$00,'K',$08,$05,$08,$05
 dc.b $00,$0B,$00,$0B,$00,$8B,$00,$8B
deada:
 dc.b $01,$10,$01,$22,$00,$E6,$00,$C6
 dc.b $00,$01,$00,$01,$00,$81,$00,$81
 dc.b 'a',$E0,$E1,$E2,$E3,$E4,$E0,$E0,$E2
 dc.b $E0,$E0,$E0,$E1,$E2,$E1,$E1,$E1,$E1
 dc.b 'A',$C0,$C1,$C2,$C3,$C4,$C0,$C0,$C2
 dc.b $C0,$C0,$C0,$C1,$C2,$C1,$C1,$C1,$C1
deade:
 dc.b $01,$10,$01,$22,$00,$A9,$00,$A9
 dc.b $00,$05,$00,$05,$00,$85,$00,$85
 dc.b 'e',$E8,$E9,$EA,'e',$EB,$E8,$E8,$EA
 dc.b $E8,$E8,$E8,$E9,$EA,$E9,$E9,$E9,$E9
 dc.b 'E',$C8,$C9,$CA,'E',$CB,$C8,$C8,$CA
 dc.b $C8,$C8,$C8,$C9,$CA,$C9,$C9,$C9,$C9
deadi:
 dc.b $01,$10,$01,$22,$00,$A1,$00,$A6
 dc.b $00,$09,$00,$09,$00,$89,$00,$89
 dc.b 'i',$EC,$ED,$EE,'i',$EF,$EC,$EC,$EE
 dc.b $EC,$EC,$EC,$ED,$EE,$ED,$ED,$ED,$ED
 dc.b 'I',$CC,$CD,$CE,'I',$CF,$CC,$CC,$CE
 dc.b $CC,$CC,$CC,$CD,$CE,$CD,$CD,$CD,$CD
deadn:
 dc.b $01,$10,$01,$22,$00,$AD,$00,$AF
 dc.b $00,$0E,$00,$0E,$00,$8E,$00,$8E
 dc.b 'n','n','n','n',$F1,'n','n','n','n'
 dc.b 'n','n','n','n','n','n','n','n','n'
 dc.b 'N','N','N','N',$D1,'N','N','N','N'
 dc.b 'N','N','N','N','N','N','N','N','N'
deado:
 dc.b $01,$10,$01,$22,$00,$F8,$00,$D8
 dc.b $00,$0F,$00,$0F,$00,$8F,$00,$8F
 dc.b 'o',$F2,$F3,$F4,$F5,$F6,$F2,$F2,$F4
 dc.b $F2,$F2,$F2,$F3,$F4,$F3,$F3,$F3,$F3
 dc.b 'O',$D2,$D3,$D4,$D5,$D6,$D2,$D2,$D4
 dc.b $D2,$D2,$D2,$D3,$D4,$D3,$D3,$D3,$D3
deadu:
 dc.b $01,$10,$01,$22,$00,$B5,$00,$B5
 dc.b $00,$15,$00,$15,$00,$95,$00,$95
 dc.b 'u',$F9,$FA,$FB,'u',$FC,$F9,$F9,$FB
 dc.b $F9,$F9,$F9,$FA,$FB,$FA,$FA,$FA,$FA
 dc.b 'U',$D9,$DA,$DB,'U',$DC,$D9,$D9,$DB
 dc.b $D9,$D9,$D9,$DA,$DB,$DA,$DA,$DA,$DA
deadz:
 dc.b $01,$10,$01,$22,$00,$B1,$00,$AC
 dc.b $00,$19,$00,$19,$00,$99,$00,$99
 dc.b 'y','y',$FD,'y','y',$FF,'y','y','y'
```

```
 dc.b 'y','y','y',$FD,$FD,$FD,$FD,$FD,$FD
 dc.b 'Y','Y',$DD,'Y','Y','Y','Y','Y','Y'
 dc.b 'Y','Y','Y',$DD,$DD,$DD,$DD,$DD,$DD
deadspace:
 dc.b $01,$04,$00,$A0
 dc.b $20,$60,$B4,$5E,$7E,$A8,$60,$60,$5E
 dc.b $60,$60,$60,$B4,$5E,$B4,$B4,$B4,$B4
strtab:
 dc.b $01,$04,$02,$05
 dc.b $09,$9B,$5A
strcdown:
 dc.b $02,$04,$02,$06
 dc.b $9B,$41,$9B,$54
strcup:
 dc.b $02,$04,$02,$06
 dc.b $9B,$42,$9B,$53
strcright:
 dc.b $02,$04,$03,$06
 dc.b $9B,$43,$9B,$20,$40
strcleft:
 dc.b $02,$04,$03,$06
 dc.b $9B,$44,$9B,$20,$41
strf1:
 dc.b $03,$04,$04,$07
 dc.b $9B,$30,$7E,$9B,$31,$30,$7E
strf2:
 dc.b $03,$04,$04,$07
 dc.b $9B,$31,$7E,$9B,$31,$31,$7E
strf3:
 dc.b $03,$04,$04,$07
 dc.b $9B,$32,$7E,$9B,$31,$32,$7E
strf4:
 dc.b $03,$04,$04,$07
 dc.b $9B,$33,$7E,$9B,$31,$33,$7E
strf5:
 dc.b $03,$04,$04,$07
 dc.b $9B,$34,$7E,$9B,$31,$34,$7E
strf6:
 dc.b $03,$04,$04,$07
 dc.b $9B,$35,$7E,$9B,$31,$35,$7E
strf7:
 dc.b $03,$04,$04,$07
 dc.b $9B,$36,$7E,$9B,$31,$36,$7E
strf8:
 dc.b $03,$04,$04,$07
 dc.b $9B,$37,$7E,$9B,$31,$37,$7E
strf9:
 dc.b $03,$04,$04,$07
 dc.b $9B,$38,$7E,$9B,$31,$38,$7E
strf10:
 dc.b $03,$04,$04,$07
 dc.b $9B,$39,$7E,$9B,$31,$39,$7E
strhelp:
 dc.b $03,$02
 dc.b $9B,$3F,$7E
name:dc.b "d",0
```

You can write your own keymap as a data block and assemble it. After linking you have a keymap file which can be installed using the CLI `SetMap` command. You define an `Exec` node in the assembler source of the keymap, which is used later to link the keymap to the `keymap.resource` structure. The `KeyMap` structure follows with all of the tables. Remember to provide the names of the `Exec` nodes, otherwise you cannot find the keymap after you have added it to the system.

Here are two more structures. The first comprises the entire structure of the keymap file, only you don't see the tables for the actual conversion because these belong to the `KeyMap` structure. The second states the structure of the `keymap.resource`. You can get this from the list of all of the resources using the `FindName` function (`execBase`), then search for a keymap of the same name with the same function.

```
struct  KeyMapNode
{
0x00  0     struct Node kn_Node;
0x0E 14     struct KeyMap kn_KeyMap;
};
struct  KeyMapResource
{
0x00  0     struct Node kr_Node;
0x0E 14     struct List kr_List;
0x1C 28
};
```

When you want to change your keymap, use the disassembled keymap listed above. Enter the listing, change the arrangement of the keys that you want to change and save this keymap under an unusual name (include an `.asm` extension if possible). For example, say you named the file `demo.asm`. You can assemble this file with:

```
as demo
```

And then link it:

```
ln demo
```

These calls are for the Aztec assembler. They should work on other assemblers. Above all, no library is linked to the keymap. The following CLI entries copies this keymap to a boot disk:

```
copy demo to Devs:Keymaps
SetMap demo
```

If you want to access this keymap, you must add it to the startup sequence of your boot disk, or from the tool (application program) that should use this keymap. This could look like the following example:

```
struct KeyMapResource *FindName();
struct KeyMapResource *kmRes;
struct KeyMap *km = Null;
.
if (kmRes = FindName(&(SysBase-
>ResourceList),"keymap.resource"))
   km = FindName(&(kmRes->kr_List),"demo");
```

This keymap (`km`) can be given either at the call of the `RawKeyConvert` function or in the `StringInfo` structure of a gadget (`AltKeyMap`). This enables you to access the new keyboard arrangement.

If you don't want to reassemble and relink the keymap every time you want to change a key arrangement, use programs that let you change the keyboard arrangement during program execution. This is helpful if you want to change the arrangement more than once. You can find many such programs available commercially as well as in the public domain and shareware world.

If you want to know the current arrangement of your keyboard, use the `KeyToy` tool usually located in the `Tools` directory of your Extras diskette (your `KeyToy` may be in another directory—check with your dealer). The Amiga 2000 `KeyToy` is called `KeyToy2000`, while the Amiga 500 `KeyToy` is called `KeyToy500`. When you start this tool, it displays an image of a keyboard on the screen, and shows which characters you get when you press the corresponding key on the real keyboard.

By clicking on the <Shift>, <Alt> or <Ctrl> keys you can see what function the corresponding key has. While this tool cannot display strings that are modified through a dead key, it shows your letters in italics. The actual dead keys are displayed in orange, and the accent indicates that this dead key was created using other keys.

To conclude, the console device functions are ready to get a pointer to the current keymap, set the current keymap and convert a RAWKEY number into an ASCII string. The last function (`RawKeyConvert`) is the most interesting of all for the programs that wait for RAWKEY input and must convert these themselves.

Appendix A: Function List

Appendix B: Bibliography

ROM Kernel Reference Manual: Libraries and Devices
Commodore-Amiga, Inc.
Addison Wesley Publishing Company, Inc.
Source for: Library functions version 1.1
Device functions 1.1

ROM Kernel Reference Manual: Exec
Commodore-Amiga, Inc.
Addison Wesley Publishing Company, Inc.
Source for: Library functions version 1.1
Structure documentation 1.1

Amiga Intuition Reference Manual
Commodore-Amiga, Inc.
Addison Wesley Publishing Company, Inc.
Source for: Library functions, Version 1.1
Structure documentation, Version 1.1

The AmigaDOS Manual (2nd Edition)
Commodore Amiga, Inc.
The Bantam Amiga Library
Source for: Library functions, Version 1.2
Structure documentation, Version 1.2

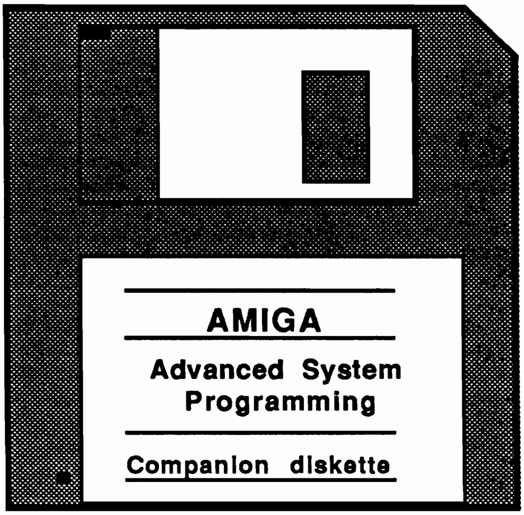
Amiga System Programmer's Guide
Abacus
Source for: Library functions Version 1.2
Structure documentation, Version 1.2

Amiga Graphics Inside and Out
Abacus
Source for: Library functions Version 1.2
Structure documentation, Version 1.2

Commodore Amiga Developers Conference Documentation
Commodore-Amiga, Inc.
Source for: Operating system information, Version 1.3

Index

Companion Diskette



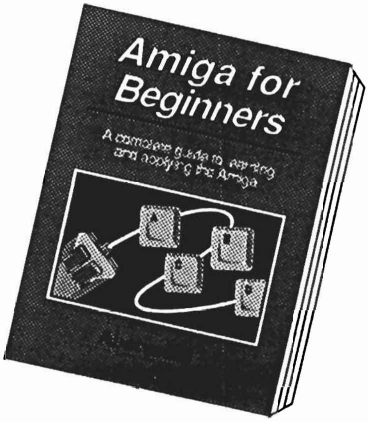
For your convenience, the program listings contained in this book are available on an Amiga formatted floppy disk. You should order the diskette if you want to use the programs, but don't want to type them in from the listings in the book.

All programs on the diskette have been fully tested. You can change the programs for your particular needs. The diskette is available for $14.95 plus $2.00 ($5.00 foreign) for postage and handling.

When ordering, please give your name and shipping address. Enclose a check, money order or credit card information. Mail your order to:

Books for the AMIGA

Amiga for Beginners



Amiga For Beginners- the first volume in our Amiga series, introduces you to Intuition (Amiga's graphic interface), the mouse, windows, the CLI, and Amiga BASIC and explains every practical aspect of the Amiga in plain English.The glossary, "first-aid" appendix, icon appendix and technical appendix are invaluable to the beginner.

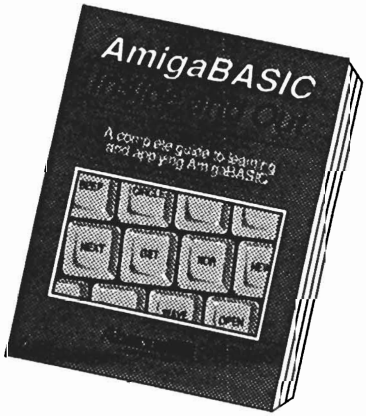
Topics include:

- Unpacking and connecting the Amiga components
- Starting up your Amiga
- Customizing the Workbench
- Exploring the Extras Disk
- Taking your first steps in the AmigaBASIC programming language
- AmigaDOS functions
- Using the CLI to perform 'housekeeping' chores
- First Aid, Keyword, Technical appendixes
- Complete set-up instructions
- Backing up important diskettes
- Setting Preferences
- Creating your own icons

No Optional Disk Available

Volume 1 Suggested retail price $16.95 ISBN 1-55755-021-2

AmigaBASIC: Inside & Out



AmigaBASIC- Inside and Out- THE definitive step-by-step guide to programming the Amiga in BASIC. Every AmigaBASIC command is fully described and detailed. Topics include charts, windows, pull down menus, files, mouse and speech commands.

Features:

- Loaded with real working programs
- Video titling for high quality object animation
- Windows
- Pull-down menus
- Moused commands
- Statistics
- Sequential and random files
- Exciting graphics demonstrations
- Powerful database
- Charting application for creating detailed pie charts and bar graphs
- Speech utility for remarkable human voice syntheses demonstrations
- Synthesizer program to create custom sound effects and music.

Volume 2 Suggested retail price $24.95 ISBN 0-916439-87-9

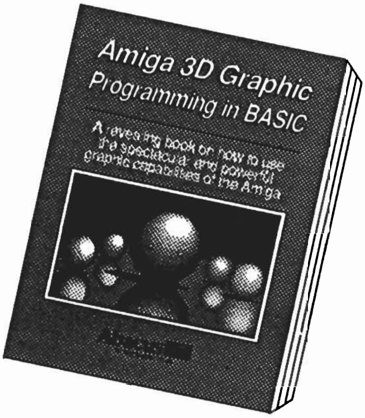
Optional Diskette $14.95 #612

Save Time and Money!-Optional program disks are available for all our Amiga reference books (except Amiga for Beginners and AmigasDOS Quick Reference). Programs listed in the book are on each respective disk and saves countless hours of typing! $14.95

Books for the AMIGA

Amiga 3-D Graphics Programming in BASIC

Shows you how to use the powerful graphics capabilities of the Amiga. Details the techniques and algorithm for writing three-dimensional graphics programs: ray tracing in all resolutions, light sources and shading, saving graphics in IFF format and more.



Topics include:

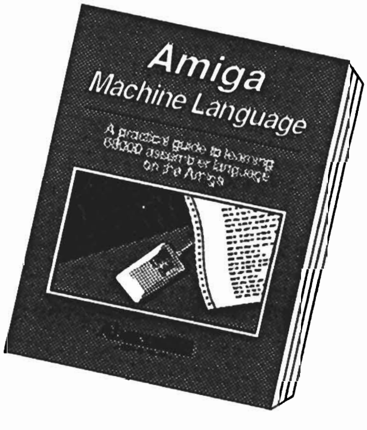
- Basics of ray tracing
- Using an object editor to enter three-dimensional objects
- Material editor for setting up materials
- Automatic computation in different resolutions
- Using any Amiga resolution (low-res, high-res, interface, HAM)
- Different light sources and any active pixel
- Save graphics in IFF format for later recall into any IFF compatible drawing program
- Mathematical basics for the non-mathematician

Volume 3 Suggested retail price $19.95 ISBN 1-55755-044-1

Optional Diskette	**$14.95**	**#677**

Amiga Machine Language

Amiga Machine Language introduces you to 68000 machine language programming presented in clear, easy to understand terms. If you're a beginner, the introduction eases you into programming right away. If you're an advance programmer, you'll discover the hidden powers of your Amiga. Learn how to access the hardware registers, use the Amiga libraries, create gadgets, work with Intuition and much more.



- 68000 address modes and instruction set
- Accessing RAM, operating system and multitasking capabilities
- Details the powerful Amiga libraries for using AmigaDOS
- Speech and sound facilities from machine language
- Simple number base conversions
- Text input and output
- Checking for special keys
- Opening CON: RAW: SER: and PRT: devices
- New directory program that doesn't access the CLI
- Menu programming explained
- Complete Intuition demonstration program including Proportional, Boolean and String gadgets.

Volume 4 Suggested retail price $19.95 ISBN 1-55755-025-5

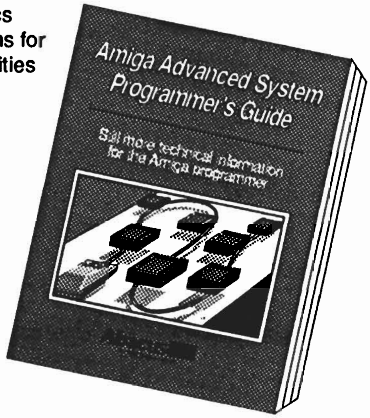
Optional Diskette	**$14.95**	**#662**

Save Time and Money!-Optional program disks are available for all our Amiga reference books (except Amiga for Beginners and AmigasDOS Quick Reference). Programs listed in the book are on each respective disk and saves countless hours of typing! $14.95

Books for the AMIGA

Advanced System Programmer's Guide

A follow up volume to the internals of the Amiga covering even more topics including Kickstart and covering Workbench 1.3. Presents the conventions for systems programming. Very thorough explainations of accessing the facilities provided by the Libraries, input and output using the Devices, using and changindg the preferences. Describes the various standard IFF formats- graphics, text music.



Topics include:

- Using the new AmigaDOS ... eferences 1.3 and the Extr...
- Doz... your pro...
- How ... Amig...
- More ... video ...
- Access ... in AmigaBASIC
- Disablir ... drives, Hardware hacking
- New dev ... ed: NewCon, PIPE and using the Mount command

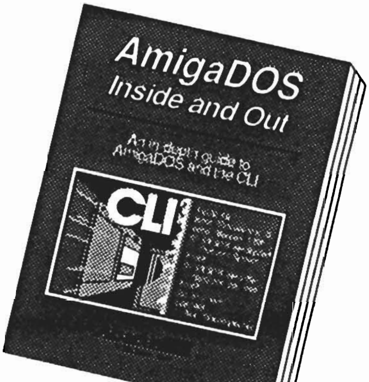
Available November 89

Volume 7 Suggested retail price $34.95 ISBN 1-55755-047-6

Optional Diskette	**$14.95**	**#697**

AmigaDOS: Inside & Out

AmigaDOS: Inside & Out-covers the insides of AmigaDOS from the internal design up to practical applications. Includes detailed reference section, tasks and handling, DOS editors ED and EDIT, how to create and use batch files, multitasking, and much more.



Topics include:

- 68000 microprocessor architecture
- AmigaDOS - Tasks and handling
- Detailed explanations of CLI commands and their functions
- DOS editors ED and EDIT
- Operating notes about the CLI (Wildcards, shortening input and output)
- Amiga devices and how the CLI uses them
- Batch files - what they are and how to write them
- Changing the startup sequence
- AmigaDOS and multitasking
- Writing your own CLI commands
- Reference to the CLI, ED and EDIT commands
- Resetting priorities - the TaskPri command
- Protecting your Amiga from unauthorized use

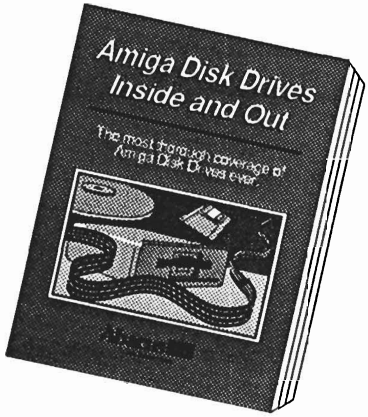
Volume 8 Suggested retail price $19.95 ISBN 1-55755-041-7

Optional Diskette	**$14.95**	**#667**

Save Time and Money!-Optional program disks are available for all our Amiga reference books (except Amiga for Beginners and AmigasDOS Quick Reference). Programs listed in the book are on each respective disk and saves countless hours of typing! $14.95

Books for the AMIGA

Amiga Disk Drives: Inside & Out



Amiga Disk Drives: Inside & Out is the most in-depth reference available covering the Amiga's disk drives. Learn how to speed up data transfer, how copy protection works, computer viruses, Workbench and the CLI DOS functions, loading, saving sequential, relative file organization, more.

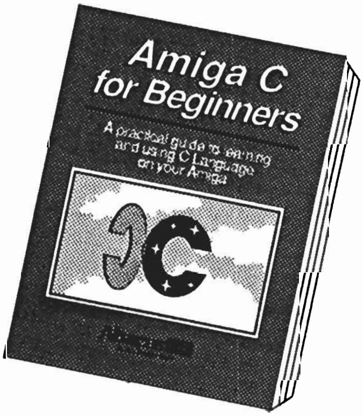
Topics include:

- Floppy disk operation from the Workbench and CLI
- BASIC: Loading, saving, sequential and relative files DOS functions
- File management: Block types, boot blocks, checksums, file headers, hashmarks and protection methods
- Viruses: Protecting your boot block
- Trackdisk.device: Commands, structures
- Trackdisk-task: Function and design
- Diskette access with DOS
- MFM, GCR, Track design, blockheader, data blocks, checksums, coding and decoding data , hardware registers, SYNC, and interrupts

Volume 9 Suggested retail price $29.95 ISBN 1-55755-042-5

Optional Diskette $14.95 #672

Amiga C for Beginners



An introduction to learning the popular C language. Explains the language elements using examples specifically geared to the Amiga. Describes C library routines, how the compiler works and more.

Topics include:

- Particulars of C
- How a compiler works
- Writing your first program
- The scope of the language (loops, conditions, functions, structures)
- Special features of C
- Important routines in the C libraries
- Input/Output
- Tricks and Tips for finding errors
- Introduction to direct programming of the operating system (windows, screens, direct text output, DOS functions)
- Using the LATTICE and AZTEC C compilers

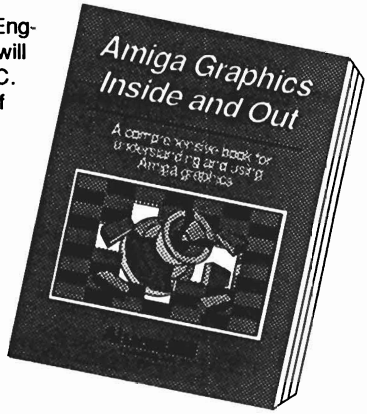
Volume 10 Suggested retail price $19.95 ISBN 1-55755-045-X

Optional Diskette $14.95 #682

Save Time and Money!-Optional program disks are available for all our Amiga reference books (except Amiga for Beginners and AmigasDOS Quick Reference). Programs listed in the book are on each respective disk and saves countless hours of typing! $14.95

Books for the AMIGA

Amiga Graphics Inside & Out



The Amiga Graphics Inside & Out book will show you simply and in plain English the super graphic features and functions of the Amiga in detail. You will learn the graphic features that can be accessed from AmigaBASIC or C. The advanced user will learn graphic programming in C with examples of points, lines, rectangles, polygons, colors and more. Amiga Graphics Inside & Out contains a complete description of the Amiga graphic system - View, ViewPort, RastPort, bitmap mapping, screens, and windows.

Topics include:

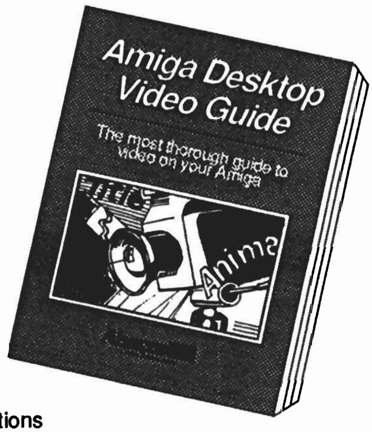
- Accessing fonts and type styles in AmigaBASIC
- CAD on a 1024 x 1024 super bitmap, Using graphic library routines
- New ways to access libraries and chips from BASIC - 4096 colors at once, color patterns, screen and window dumps to printer
- Graphic programming in C - points, lines, rectangles, polygons,colors
- Amiga animation explained including sprites, bobs and AnimObs, Copper and blitter programming

Volume 13 Suggested retail price $34.95 ISBN 1-55755-052-2

Optional Diskette	**$14.95**	**#727**

Amiga Desktop Video Guide

The **Amiga Desktop Video Guide** is the most complete and useful guide to desktop video on the Amiga.



Amiga Desktop Video Guide covers all the basics – defining video terms, selecting genlocks, digitizers, scanners, VCRs, camera and connecting them to the Amiga.

Just a few of the topics you'll find described in this excellent book:

- The Basics of Video
- Genlocks
- Digitizers and Scanners
- Frame Grabbers/Frame Buffers
- How to connect VCRs, VTRs, and Cameras to the Amiga
- Animation
- Video Titling
- Music and Videos
- Home Video
- Advanced Techniques
- Using the Amiga to add or incorporate Special Effects to a video
- Tips on Paint, Ray Tracing, and 3-D Rendering in Commercial Applications

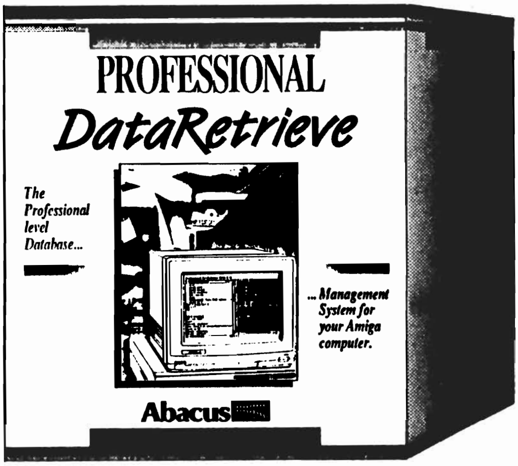
Volume 14 • Suggested Retail Price $19.95 • ISBN 1-55755-057-3

Save Time and Money!-Optional program disks are available for all our Amiga reference books (except Amiga for Beginners and AmigasDOS Quick Reference). Programs listed in the book are on each respective disk and saves countless hours of typing! $14.95

Abacus Products for *Amiga* computers

Professional DataRetrieve

The Professional Level Database Management System



Professional DataRetrieve, for the Amiga 500/1000/2000, is a friendly easy-to-operate professional level data management package with the features most wanted in a relational data base system.
Professional DataRetrieve has complete relational data mangagement capabilities. Define relationships between different files (one to one, one to many, many to many). Change relations without file reorganization.

Professional DataRetrieve includes an extensive programming laguage which includes more than 200 BASIC-like commands and functions and integrated program editor. Design custom user interfaces with pulldown menus, icon selection, window activation and more.
Professional DataRetrieve can perform calculations and searches using complex mathematical comparisons using over 80 functions and constants.

Professional DataRetrieve is a friendly, easy to operate programmable RELATIONAL data base system. **PDR** includes PROFIL, a programming language similar to BASIC. You can open and edit up to 8 files simultaneously and the size of your data fields, records and files are limited only by your memory and disk storage. You have complete interrelation between files which can include IFF graphics. NOT COPY PROTECTED. ISBN 1-55755-048-4

MORE features of **Professional DataRetrieve**

Easily import data from other databases....file compatible with standard **DataRetrieve**....supports multitasking...design your own custom forms with the completely integrated printer mask editor....includes PROFIL programming language that allows the programmer to custom tailor his database requirements...

MORE features of PROFIL include:

Open Amiga devices including the console, printer, serial and the CLI.
Create your own programmable requestors
Complete error trapping.
Built-in compiler and much, much more.

Suggested retail price: $295.00

Features

- Up to 8 files can be edited simultaneously
- Maximum size of a data field 32,000 characters (text fields only)
- Maximum number of data fields limited by RAM
- Maximum record size of 64,000 characters
- Maximum number of records disk dependent (2,000,000,000 maximum)
- Up to 80 index fields per file
- Up to 6 field types - Text, Date, Time, Numeric, IFF, Choice
- Unlimited number of searches and subrange criteria
- Integrated list editor and full-page printer mask editor
- Index accuracy selectable from 1-999 characters
- Multiple file masks on-screen
- Easily create/edit on-screen masks for one or many files
- User-programmable pulldown menus
- Operate the program from the mouse or the key board
- Calculation fields, Data Fields IFF Graphics supported
- Mass-storage-oriented file organization
- Not Copy Protected, NO DONGLE; can be in stalled on your hard drive

Advanced System Programmer's Guide for the Amiga®

Advanced System Programmer's Guide for the Amiga–
is the second comprehensive volume describing the "internals" of the Amiga. This book includes the latest information on Kickstart and Workbench 1.3. If you work with the Amiga often, you'll quickly see how helpful this book will be in uncovering important information that you may need quickly.

Programmers like yourself have asked for the information contained in ***Advanced System Programmer's Guide for the Amiga***.
If you use the libraries and devices or want to get down to the detailed levels of programming, this book will increase your understanding of the Amiga.

Optional program diskette available:
contains all of the programs listed in the book - complete, error-free and ready to run! Saves you hours of keying program listings.

Still more essential information for the Amiga programmer

Some of the topics covered include:

- Interfaces - audio, video, RGB, Centronics, serial, disk access, expansion port, keyboard
- Programming hardware - memory organization, interrupts, the Copper, Blitter and disk controller
- EXEC structures - Node, List, Libraries and Tasks
- Multitasking - Task switching, intertask communication, exceptions, traps and memory management
- I/O - device handling and requests
- DOS libraries - functions, parameters and error messages
- CLI - detailed internal design descriptions
- Devices - Trackdisk, Console, Narrator, SER, PAR, PRT and gameport

ISBN 1-55755-047-6
90000
9 781557 550477

Abacus
5370 52nd Street SE • Grand Rapids, MI 49512