AMIGA WORLD

Official AmigaDOS 2 Companion

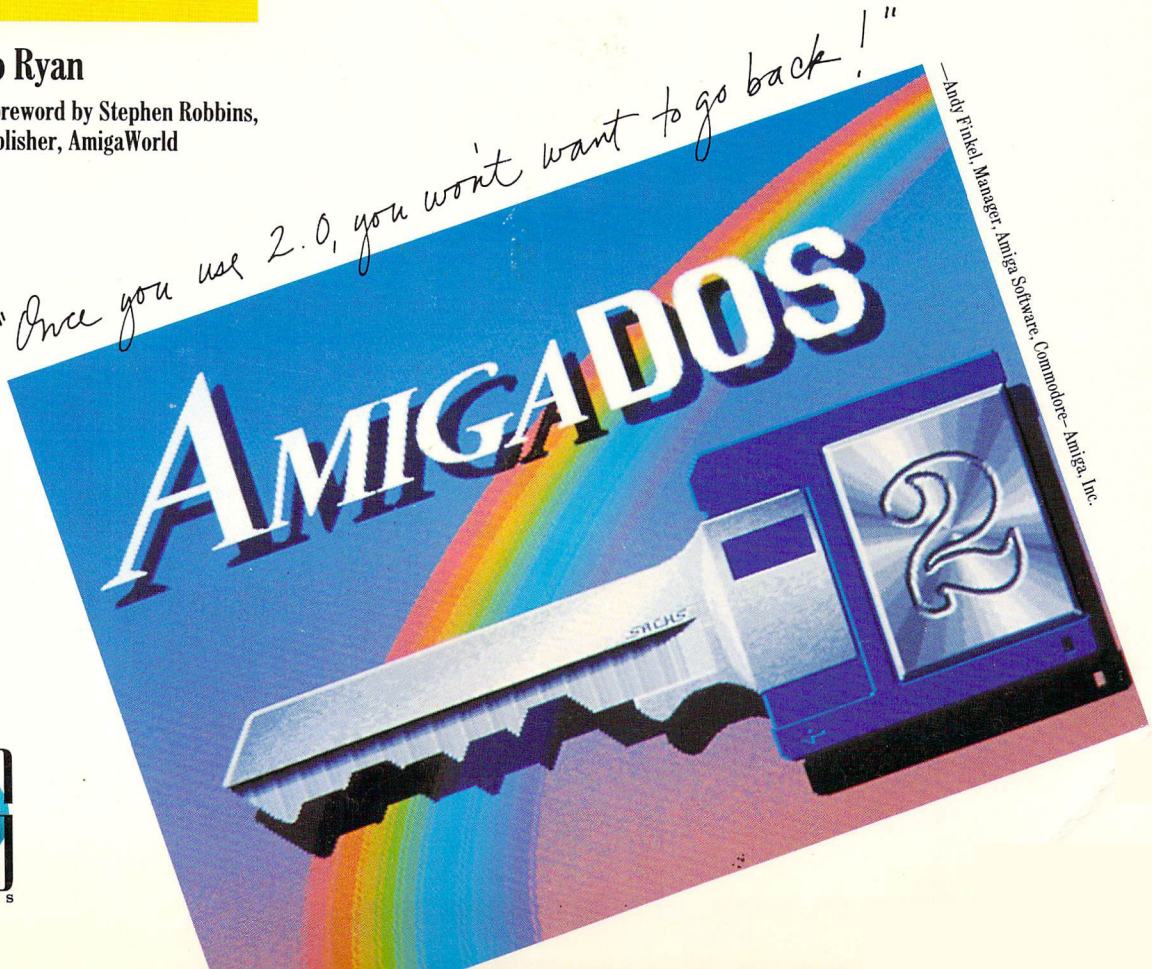
Complete guide to using the new AmigaDOS 2

Definitive reference for Amiga 3000, 2000 & 500 machines

Plus: Workbench menus & tools, Preferences, Commodities Exchange, Extras, DOS & the Shell, ARexx... and more!

by Bob Ryan

Special Foreword by Stephen Robbins, Group Publisher, AmigaWorld



IDG BOOKS

Official AmigaDOS 2 Companion

By Bob Ryan

Special Preface by
Stephen C. Robbins
Publisher
AmigaWorld Magazine

IDG Books Worldwide
San Mateo, California 94402

AmigaWorld Official AmigaDOS 2 Companion

Published by
IDG Books Worldwide, Inc.
155 Bovet Road, Suite 730
San Mateo, CA 94402
(415) 358-1250

Library of Congress Catalog Card No.: 90-84500

ISBN 1-878058-09-6

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Editor-in-Chief: Michael E. McCarthy
Associate Editor: Jeremy Judson
Production Manager: Lana Olson
Edited by Linda L.B. Laflamme, Editor, AmigaWorld Tech Journal
Interior design by Bill Hartman
Production by Hartman Publishing

Distributed in the United States by IDG Books Worldwide.
Distributed in Canada by Macmillan of Canada, a Division of Canada Publishing Corporation.
For information on translations and availability in other countries, contact IDG Books Worldwide.
For sales inquiries and special prices for bulk quantities, write to the address above or call IDG Books Worldwide at (415) 358-1250.

Dedication

To Rosemary Gibson Ryan and Hannah Rose Ryan, with all my love.

Acknowledgements

I would like to thank Linda J. B. Laflamme, my friend and editor, for making this book as intelligible as it is. Linda made me a better writer. I'd also like to thank my friends Dan Sullivan, for pitching in on Chapter 7, and Lou Wallace, for the general commiseration that only a fellow author under deadline can supply.

Speaking of deadlines, I'd like to acknowledge the patient help and understanding of Mike McCarthy of IDG Books. Mike and the rest of the staff at IDG Books have been a pleasure to work with.

I would like to thank the many folks at Commodore who answered my questions about Amiga OS 2.0 and to acknowledge the help of Keith Masavage, formerly Amiga Product Manager, in keeping me up to date with the latest versions of the software.

Finally, I would like to thank my family and friends — especially my wife Rose and daughter Hannah — for the sacrifices they've made so that I could write this book. I'm back!

(The publisher would like to thank especially Bill Murphy, without whom this book would not have been possible.)

Table of Contents

Foreword

By Stephen C. Robbins
Publisher
AmigaWorld Magazine

In 1985, I had the privilege of using a new computer that was soon to be marketed by Commodore Business Machines. Although it was uncompleted, the potential of the Amiga was evident, encompassing true multitasking, 4,096 colors, a graphical interface, Motorola 68000 technology with custom co-processing chips, the potential of MS-DOS compatibility, and (the most attractive feature of all) a street price of under $1,000. We were sure this computer was going to be a winner.

Almost five years later, we still feel the same. Today, the Amiga is a well-respected platform in the computing industry, still delivering one of the best values when comparing price to performance. The Amiga can be found in Walt Disney Studios being used by professional animators, in Orlando at the Universal Studios controlling earthquake-simulation machinery, at the SuperBowl lighting up the sign machine, and in many cable television stations across the country running weather charts, handling titling and special effects, and displaying program information. More importantly, the Amiga has become accepted as the premiere platform in the video industry due to its NTSC compatibility, as well as low-cost, high-quality hardware and software options.

Yes, the Amiga has come a long way, and *AmigaWorld Magazine* has been there from the beginning, covering all the developments within the marketplace. We launched *AmigaWorld* because of our strong belief in the successful future of the Amiga. We have reported on the market, reviewed products, featured unique and interesting applications, and as a consequence are considered the foremost authority in the Amiga arena. Yet, most important of all, we have taught users how to achieve maximum results with their computers through tutorial information.

In the beginning, due to the Amiga's proprietary operating system and the lack of understandable sources of information on its use, the majority of letters sent to the offices of *AmigaWorld* pertained to the use of AmigaDOS. In 1988, we published our first book, *The Amiga Companion,* in response to that need. *The Amiga Companion* has sold over ten thousand copies to date. With changes to the Amiga's custom chips and the release of a new operating system, Amiga OS 2.0, the need for a new, more complete source of information has become very evident. This volume, *AmigaWorld Official AmigaDOS 2 Companion,* was

born out of the need for an easy path to follow through the maze of learning a new operating system and all its attendant features and benefits.

AmigaWorld Official AmigaDOS 2 Companion will guide both the advanced and the fledgling user of the Amiga through all aspects of the Workbench, which has been enhanced to include new capabilities and functionalities for the advanced user. It will provide an overview of how the Amiga works and how to get the most from the faster, better-performing interface made possible by the Amiga OS 2.0. ARexx, the built-in support for communication and control of outside devices and software, is also covered. Most crucial of all, the book points out inaccuracies in the documentation included with Amiga OS 2.0.

As users of the Amiga ourselves, we know *AmigaWorld Official AmigaDOS 2 Companion,* will become an indispensable guide for attaining the best performance from your Amiga. The book should be kept next to your computer at all times, ready to help whenever needed through crashes and system errors. (There are no more gurus in 2.0.) Although the Amiga and its operating system have come a long way since 1985, this book is essential to mastering the computer successfully.

Due to recent changes in the Amiga, I am confident of the continued success of the computer we have all been enjoying for five years. The entire staff of *AmigaWorld Magazine* hopes that, through the continued publication of our magazine and by offering special products such as this book and our videotapes, we will be able to increase your enjoyment, productivity, and creativity with the Amiga computer. We look forward to your continued satisfaction with the services we provide.

Introduction

Read Me First

Don't you just love those "read-me" files that so many software developers include on their release disks? Whenever I see one, I let out a groan, because it invariably means that something has been left out of the documentation. I really shouldn't complain, however, because without read-me files I wouldn't be able to use the software fully.

In fact, after using Workbench and Kickstart 2.0 for a while, I actually started to yearn for some read-me files. The *Using the System Software* manual that comes with Amiga OS 2.0 was completed before the software was finished, so it contains a lot of mistakes and omissions. I've come to look upon our book, the *AmigaWorld Official AmigaDOS 2 Companion*, as a giant, printed read-me file for Amiga OS 2.0. I hope you will too.

Some Do's and Don'ts

The philosophy behind this book is that the best way to learn about your Amiga computer is to use it. Consequently, I present a lot of examples in this book that I hope you will follow with your computer. This book is meant to be read as you work at your Amiga.

Before starting out with Amiga OS 2.0, you should make backups of all your important system disks so that you can reconstitute your system in case something goes wrong. You learn how to copy disks in Chapter 2 (the drag method) and Chapter 3 (the menu method).

Most of the examples in the book involve disk activity of one kind or another. It is critically important that you do not eject a disk while the disk-activity light is on! Always wait at least a couple of seconds after the light goes off before ejecting a disk. If you do not, you could corrupt the contents of the disk.

One more thing about examples: Many of them, particularly in the AmigaDOS section (Chapters 7-10), require that you enter a control character. You enter such a character by pressing the Control key and the character key at the same time. In the book, I use the abbreviation CTRL for the Control key. When you see the notation CTRL-C, you should hold the Control key down and press the C key.

Jumping In

Chapter 1 provides a general introduction to computer operating systems, the Amiga operating system, and user interfaces. It isn't essential to understanding what your Amiga does, so you can defer reading it to a later time if you want to jump right in. Chapters 2 through 6 cover the Workbench interface. This is a departure from *The Amiga Companion*, which only covered the Shell. I feel that, with the improvements made to Workbench and Intuition, this expanded coverage is essential. Workbench is now a viable alternative to the Shell for all types of users, even experts.

Chapters 7 through 10 document the Shell interface and the AmigaDOS commands. They provide plenty of opportunities for hands-on experimentation. Chapters 11 and 12 provide an introduction to ARexx. This fascinating language is what multitasking is all about. Because of the sheer impossibility of documenting ARexx in two chapters (it needs a book the length of this one), I opted to make Chapter 11 an example-strewn introduction to the language and use Chapter 12 to present a fully annotated program that uses the ARexx interprocess communications features. I like to think that Chapters 11 and 12 give you the information you need to tackle Commodore's murky documentation on the subject.

Following Tradition

This book owes much, especially in its hands-on philosophy, to its predecessor, AmigaWorld's *The Amiga Companion* by Rob Peck, which covered AmigaDOS 1.3. I was the primary editor of that book, and I hope that this one does justice to the standards set by Rob in his work. Rob Peck died this past summer; he is missed both for the person he was and for the enthusiasm he brought to the Amiga community.

In the introduction to *The Amiga Companion*, Rob invited readers to write him with questions and problems. I extend the same invitation to you. Drop me a letter with a self-addressed stamped envelope care of IDG Books. I don't promise to answer every letter, but I'll do my best.

Bob Ryan
New Ipswich, NH
November, 1990

– 1 –

Introducing Amiga OS

When Commodore introduced the original Amiga computer, the Amiga 1000, in the summer of 1985, most people focused on its stunning graphics. Its 4,096 colors were a welcome change at a time when all Macintoshes had only black-and-white displays, and most IBM PCs and compatibles provided either text-only or limited CGA (color graphics adapter) displays.

More surprising than the graphics, however, was the fact that the Amiga could run more than one program at a time. Called *multitasking*, this was a standard feature of mainframe and minicomputer systems: It was unheard of on a desktop computer.

So unique was the Amiga's multitasking, and so amazing that Commodore was able to pull it off in a machine with as little as 256K bytes of memory, that many people simply didn't believe it. (By contrast, IBM and Microsoft recommended a minimum of 3,000K bytes — that's three million bytes — of memory to run OS/2, their multitasking operating system.) For a long time after the Amiga was introduced, you could still run into industry skeptics who discounted Amiga multitasking as "not true multitasking." They seemed unwilling to believe that a computer that sold for $1,200 dollars could have an operating system far in advance of anything available for a Macintosh or an IBM PC/AT.

Although the Amiga hardware — the chips and boards that make up your computer — provides important support for multitasking, it is the operating system — Amiga OS — that ultimately lets you run more than one program at a time. This book, in conjunction with the documentation you received with your computer or with the Amiga OS 2.0 enhancement kit, will explain the workings of version 2.0 of Amiga OS and show you how you can get the most out of your Amiga system.

Amiga OS Defined

As you'd expect, the *OS* in Amiga OS stands for *operating system.* Amiga OS contains the programs and routines that let you or your applications control your computer hardware. Amiga OS controls all aspects of your machine's operation, from the painting of pixels on your monitor to the allocation of memory among different programs.

Because of the precedent set in the IBM world, where the predominant operating system is MS-DOS, users have taken to referring to the entire Amiga operating system as AmigaDOS. This is technically incorrect, although, as the title of this book demonstrates, this usage is now common in the Amiga community. Unlike MS-DOS, which controls all aspects of the system, AmigaDOS refers specifically to the parts of the operating system that control disk drives and other external devices. To keep things clear, I will use AmigaDOS to refer to only a specific subset of Amiga OS throughout this book.

Amiga OS, like all software, is intangible. You can't see software or hold it in your hand — although you can hold a disk that stores software code. The best you can do is see the effects of software — the picture a graphics program creates on your monitor or the letter a word processor outputs to your printer. Software consists of instructions that your hardware can understand. Consequently, to understand Amiga OS, you also need some understanding of the hardware it controls.

The Amiga Computer

Like all computers, every Amiga model consists of a central processing unit (CPU), memory, and input/output (I/O) systems that let you get programs and data into and out of the computer (see Figure 1-1). The CPU understands hundreds of very primitive commands, such as "add what is in location 16 to what is in location 1808 and store the results in location 500" and "move the contents of location 3486 to location 890534." What gives a computer its power, therefore, is not these simple basic operations that it performs, but the speed at which it performs them. Moving a bunch of bits around in memory may not seem like a big deal, but when the result is a stunning 3-D animation, you begin to get a feeling for the sophistication possible by executing several million of these primitive instructions every second.

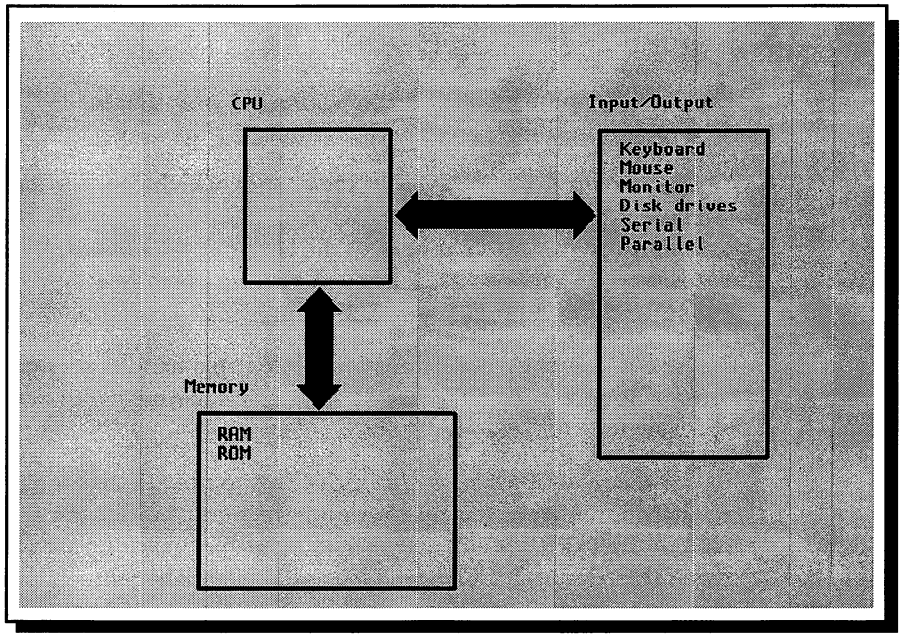


Figure 1-1 Generic Computer System

As is every computer system, the Amiga is composed of three primary subsystems: the CPU, memory, and an I/O scheme. The "brains" of a computer, the CPU is the device that executes the instructions of both the system software and applications software. In the Amiga, as in all personal computers, the CPU consists of a single chip.

Memory in a personal computer is referred to as RAM, which stands for random-access memory. RAM is a temporary storage place for programs and data. When you run a program, it loads into RAM, from where its individual instructions are fetched and executed by the CPU. Data used by the program also must be stored in RAM before the CPU can access it.

The outside world communicates with the CPU and RAM via the I/O (input/output) system. I/O devices include the keyboard, video display, mouse, disk drives, and serial and parallel ports. For example, to run a program under Workbench, you double-click on its icon. This creates a message to the operating system software, which of course executes on the CPU, to load the program into memory from the disk drive. Both the message from the mouse and the loading of the program from disk are examples of I/O functions.

The Amiga hardware differs from other personal computers in one fundamental way — the Amiga supplements the CPU's operation with a set of custom chips that perform certain functions better than the CPU. The custom chips not only speed up the tasks they assume from the CPU, they also leave the

CPU free to do more of the operations for which it is best suited. The result is a system that performs far faster than one that relies upon the CPU alone.

The Amiga custom chips are called Agnus, Denise, and Paula (see Figure 1-2). (The names are a little programmer whimsey.)

Agnus contains two coprocessors: the copper, which controls the video display, and the blitter, which excels at moving chunks of memory around. (Jay Miner, the chip's designer, used to refer to the blitter as the *bimmer*, for bit-map manipulator. The name never caught on, however, because it had already been appropriated by yuppies to describe their favorite automobile.)

Denise controls the graphics output, while Paula handles sound output and floppy disk access. Together, these chips give the Amiga much of its speed, especially in graphics applications.

One of the major functions of Amiga OS is to provide access to all the specialized hardware that makes up an Amiga. To put it simply, the Amiga hardware is so complex that few programmers are capable of writing software that controls it directly. Amiga OS provides a set of standard routines that programmers use to control the Amiga hardware. In effect, when you write a program for the Amiga, you don't program the hardware; rather, you program the system software. Only Commodore's systems programmers need know how to program the hardware directly.

Handling Multitasking

In addition to providing the routines that let programs control the hardware, Amiga OS provides routines that keep programs from stepping on one another's toes. Remember, the multitasking Amiga can run two or more programs at once. You can, for example, write a letter while your telecommunications package downloads your messages from MCI Mail and your spreadsheet recalculates your monthly budget. Amiga OS must keep track of these programs, making sure they all share properly the Amiga's hardware resources (see Figure 1-3). If they don't, the results can be disastrous.

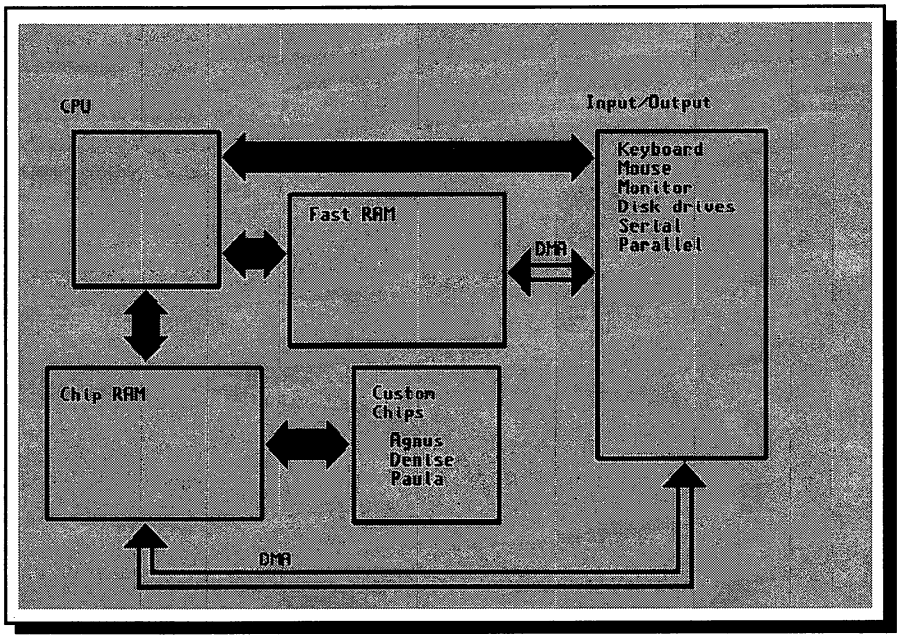


Figure 1-2 Amiga Simplified Block Diagram

The Amiga is unique among personal computers in its heavy reliance upon fast, sophisticated custom chips that enhance system performance by taking some of the processing load off of the CPU. A diagram of the Amiga shows the three custom chips. The advantage of these chips is that they are customized to perform specific functions. CPU chips by contrast are more general-purpose devices.

Note that the Amiga also has a different memory system than do other computers. Memory on the Amiga comes in two types: fast RAM and chip RAM. Chip RAM is accessible by both the CPU and the custom chips. Any data that the custom chips work on, such as graphics pictures and musical notes, must reside in chip RAM. Fast RAM, which is memory accessible only to the CPU, is where the operating system stores programs and data that do not have to be in chip RAM. The term fast RAM refers to the fact that the CPU always has access to this memory, without having to wait for the other chips to finish. In contrast, the custom chips can sometimes lock the CPU out of chip RAM while they are performing their functions.

The Amiga's I/O system is also designed for speed. While most computer systems shuttle all I/O through the CPU, the Amiga has dedicated pathways for direct-memory access (DMA), which lets I/O devices access memory with very little help from the CPU. As do the custom chips, this frees the CPU for more important tasks.

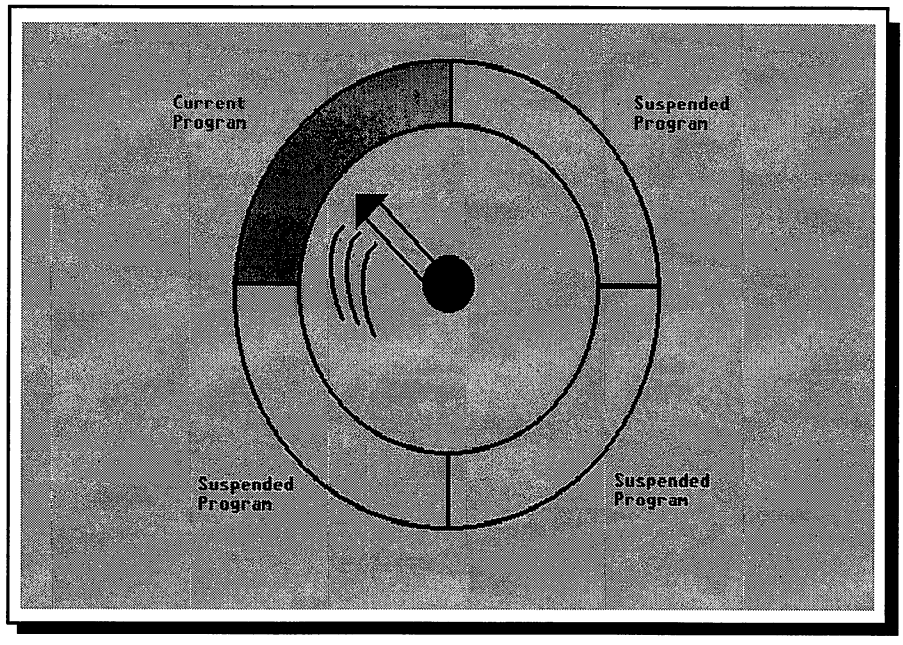


Figure 1-3 Multitasking in Action

Multitasking is commonly described as the ability to run multiple programs at one time. For practical purposes, this description is correct; technically, however, it is a total lie.

With the exception of exotic machines with parallel processing abilities, every CPU is capable of executing only one instruction at a time. That instruction, of course, can come from only one program. So, what is multitasking?

Multitasking is the ability of an operating system to switch so quickly between programs that they all appear to be executing simultaneously. The operating system actually partitions time on the CPU for the different programs in memory. When a program reaches the end of its allotted time or when it must wait for a relatively slow event such as a disk access, its status is saved by the operating system and it is put into limbo until its next turn on the CPU.

Consider, for example, two programs that need to access a printer connected to the Amiga's serial port. The first program follows all the Amiga protocols and uses Amiga OS functions to gain control of the port. Amiga OS puts a lock on the port to keep other programs from accessing it while the first program is printing. If, however, the second program doesn't go through Amiga OS, and instead writes directly to the serial port hardware, it never learns of the Amiga OS lock and will scramble the output of the first program, as well as

its own, and will probably crash the system. If the second program had been written correctly to access the serial port only through Amiga OS, it would have been locked out of the port until the first program finished, avoiding a disastrous collision. Amiga OS thus provides the means to prevent collisions between two programs that need to access the same hardware resources. It is the traffic cop of the Amiga (see Figure 1-4).

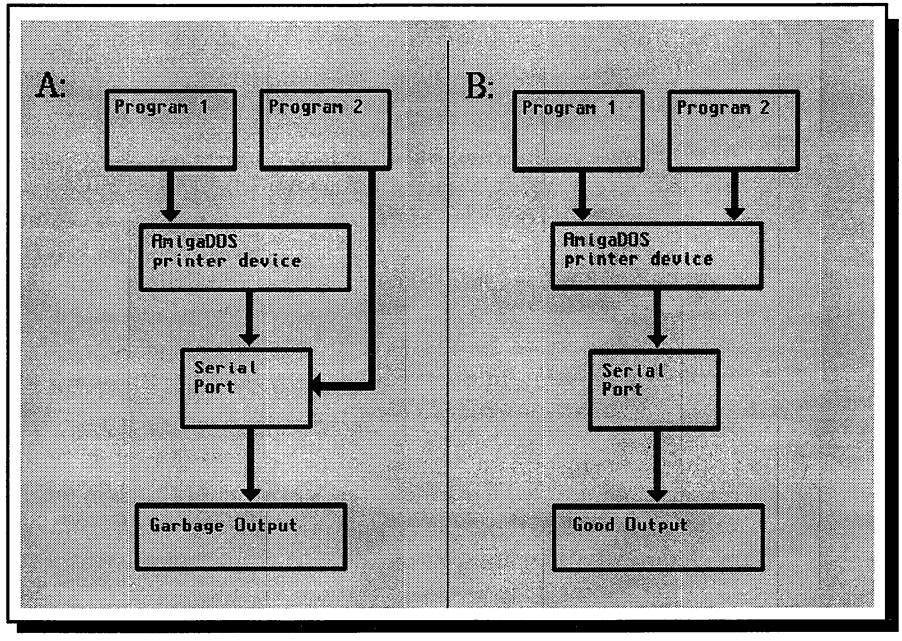


Figure 1-4 Resource Sharing

One of the more complicated jobs of a multitasking operating system is letting programs share resources gracefully. When two programs want access to the same resource, the operating system must arbitrate.

In (A), two programs need to access the printer connected to the Amiga serial port. Program 1 uses AmigaDOS to access the printer device and acquire use of the port. Program 2, contrary to the rules of Amiga programming, writes directly to the serial port registers. The result is a disaster.

In (B), Program 2 is a well-behaved application and attempts to access the printer through AmigaDOS. When it finds that another program has a lock on the port, it waits patiently until the port is free and then outputs its data to the printer. The result is the graceful sharing of a resource between two programs.

The Structure of Amiga OS

The hardware resources that Amiga OS must control include memory, the serial and parallel ports, floppy disks, the sound channels, expansion hardware (such as hard-disk drives), and the CPU itself. Amiga OS controls input from the keyboard and mouse by channeling it to the appropriate program and controls the machine's basic display output through the custom chips. Because writing one program to take care of all these details would be very difficult, Commodore has divided Amiga OS into manageable fragments called libraries and organized these libraries into a layered hierarchy.

At the lowest level — closest to the hardware — is Exec, made up of routines that talk most directly with the Amiga hardware. Exec is the master controller on the Amiga. It controls multitasking, manages the applications programs, and handles communication between different programs. On top of Exec are such things as the graphics library, used to access the Amiga graphics, the DOS library, used to access disk drives and other peripherals, and the Intuition library, used to create the Workbench interface. At the top of the hierarchy — furthest removed from the hardware — are the applications programs.

Workbench, the Shell, and ARexx are applications programs. They sit on top of the hierarchy of Amiga system software (see Figure 1-5). Unlike other applications programs, which let you perform functions such as writing a letter or painting a picture, these three applications let you control and automate the operation of your computer. While other applications are designed to meet a need in the outside world, Workbench, the Shell, and ARexx are designed to help you control the world inside your computer. They provide an interface to the functions of Amiga OS.

Amiga OS and You

Why do you need to access the functions of Amiga OS? The simple answer is to have greater control over your computer and get more work done faster. Specifically, you can break the functions you'll want to perform with Amiga OS into three categories:

- Managing files
- Configuring your system
- Automating tasks

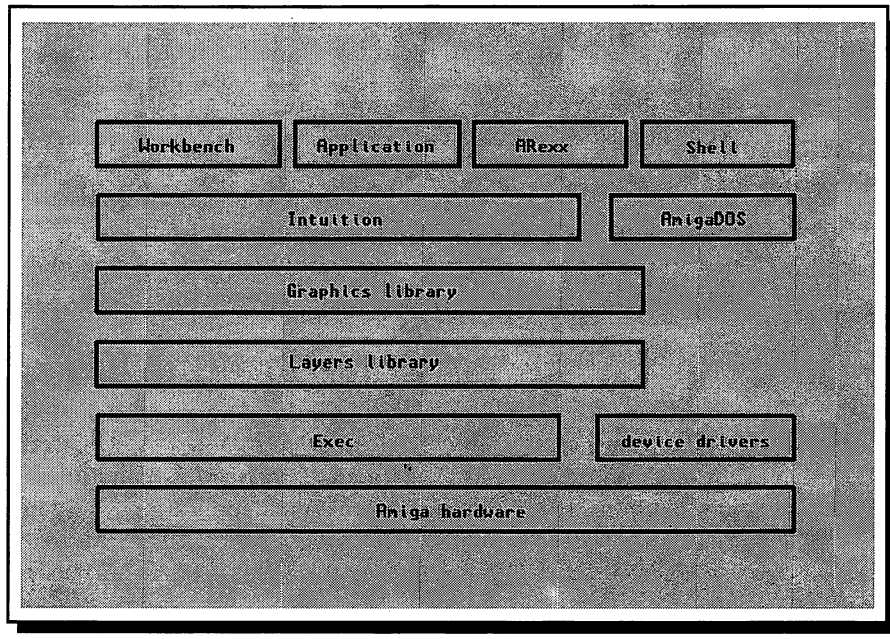


Figure 1-5 Structure of Amiga OS

Amiga OS uses layers to insulate both programmers and users from the complexities of the Amiga system. The lowest level of system hardware are Exec and the software devices, which communicate directly with the hardware. Above Exec are the special function libraries such as layers.library, graphics.library, intuition.library, and dos.library. Applications sit on top of the libraries and gain access to the Amiga hardware through them.

The layers of libraries provide different levels of abstraction to Amiga programmers and users. When using Workbench, you are actually manipulating features of the Intuition and DOS libraries, and through them the layers, graphics, and Exec libraries that talk directly to the hardware. But you don't have to know anything about all these layers in order to access their functions. The same is true for programmers. If a simple upper layer like the DOS library provides the functions they need, they don't have to delve into the arcane world of Exec and the hardware device controllers. As in a house, these layers provide insulation.

File Management: This is the most basic and important task you can perform with Amiga OS. Everything you do with a computer involves files. When you run a program, you are actually instructing AmigaDOS to load the program file from disk and execute it. When you save a picture file or an animation, you are creating a disk file. Files are how computers store programs and data. You have to know how to manage files; how to store them, copy them, move

them around, and back them up. Without knowing how to do these things you could wipe out a year's worth of work in a few seconds.

System Configuration: Your Amiga gives you a bewildering array of options for customizing your system. Using Amiga OS, you can select your printer and control aspects of its output, change the fonts used by the system, even control the output resolution of your display. As you learn more about the system, you can experiment with different configurations to come up with the one that best suits your requirements and tastes. Once you settle on one, you'll be amazed at how such a personalized environment makes you a far more efficient user.

Task Automation: Amiga OS provides a number of ways to automate certain tasks. The Shell, for example, lets you string together commands you would normally enter one at a time. ARexx is even more powerful in this regard. It is a programming language that gives you a lot of flexibility in automating Amiga OS functions. It also lets you create "super applications" by combining the functions of different applications programs that support it. ARexx is certainly more complex than Workbench or the Shell — it can also be far more powerful.

The chapters that follow examine how you can manage files, configure the system, and automate tasks with each interface available. Which interface you choose depends upon what you need to do and how much control you want to have over the system.

The Interface Question

Back in the early '70s, the dark ages of mainframes and batch processing, the interface to a computer was usually a punch-card machine. When I was in college, I'd write a program in long hand, type each line onto a punch card, and take the entire deck to a bin where it sat until one of the computer operators ran my program. Normally, I had to wait overnight for the results of my program. Consequently, just weeding the typing mistakes out of my card deck usually took me a couple of weeks . Obviously, punch-card machines aren't the best way to get a computer to do what you want.

With the proliferation of minicomputers in the mid-'70s, punch-cards were replaced by video display terminals (VDTs). These hooked you directly to a large computer and let you work interactively with it. You could run programs and manage files in real time. VDTs were an enormous improvement over punch cards.

When personal computers became popular in the late '70s and early '80s, they used an interface similar to those of minicomputers. Like VDTs, early personal computer displays were text-only. You typed in a command, and the computer executed it. You still had to know the syntax of the commands and be a fairly good typist.

During the 1970s, however, a group of people at the Xerox Palo Alto Research Center (PARC) had a better idea. They wanted to make computers and computing accessible to more people. Their research resulted in the development of the Xerox Star, a graphics-based workstation that featured windows, menus, and a mouse as a pointing device. The Star didn't make a very big splash commercially, but it did make an impression on the folks at Apple, who incorporated elements of its interface into the ill-fated Lisa and then the Macintosh. By the time the Amiga was launched, such easy-to-use, graphical-user interfaces (GUIs) were all the rage.

In 1985, the Amiga was unique among personal computers in that it offered both a VDT-style command-line interface (CLI) and a Macintosh-style GUI. Although the original CLI has since been replaced by the Shell and the current Workbench (version 2.0) bears little resemblance to the original, Commodore has remained steadfast in its commitment to giving you a choice as to how you would like to work with your computer. Both interfaces have a lot going for them.

The Workbench Approach

As a GUI, the primary aim of the Workbench is to give you an easy-to-use interface that is consistent across different applications. The key here is consistency.

In the old days of personal computing, every program had its own interface. That is, every program had a unique way of letting you get at its functions. For example, a word processor might require that you enter CTRL-L to access the part of the program that loads a file, while your spreadsheet might use ESC-D, to bring up a menu of disk functions, from which you would choose Load. Early personal computers such as the Apple II and the IBM PC had no standard way of giving you or a programmer access to their functions. Thus, no two programs worked alike, creating a lot of confusion. If you used four or five different applications regularly, you had to know four or five different sets of commands. If you had some programs you didn't use often, you were constantly fishing through manuals refreshing your memory about their command structures.

The aim of graphical interfaces such as Workbench is to end all that. By using the Intuition library that Workbench is built upon, programmers automatically ensure that their programs share a similar look with other Amiga programs. Moreover, pull-down menus safeguard you from having to learn a lot of obtuse commands to use a program. In fact, if a program has a well-constructed menu structure, you may not even have to read the manual to use the program. In effect, the menus guide you through the program's operation. A menu item labeled "Load File" is a lot clearer than CTRL-L.

While Workbench has always been an easy-to-use interface, it has lacked the consistency across applications that characterizes programs running under the Macintosh Desktop interface. Commodore is as much to blame as the software developers. When the Amiga was released, Commodore didn't supply many of the standard tools, such as a file requester, that are available to programmers on the Mac. Neither did Commodore promulgate and enforce a strict set of style guidelines for Amiga programs. As a result, early Amiga programs used the Workbench interface in a variety of ways.

Workbench 2.0, the Workbench component of Amiga OS 2.0, addresses this consistency problem. With it, Commodore has given programmers a richer set of standard tools, such as standard file and font requesters. (You'll learn more about these in the next chapter.) The company has also written new interface guidelines for developers and given Workbench a beautiful new look. The old Workbench was so ugly, with its flat, chunky graphics that many developers didn't want to use it. Workbench 2.0 will attract developers because it will make their programs look much better.

Given that Workbench is so easy to use, you might wonder why Commodore bothers to provide other interfaces to Amiga OS. The answer is that Workbench isn't as flexible or as powerful as the Shell or ARexx. Simplicity and consistency has a price, which Workbench pays by being less powerful than the other Amiga OS interfaces.

Shell Around AmigaDOS

The Amiga version of a command-line interface, the Shell, is a program that lets you execute AmigaDOS commands by entering them from your keyboard. The Shell is the second generation Amiga command-line interface. The first was the CLI, which came with all versions of Amiga OS through version 1.3. (The CLI on Workbench 2.0 is synonymous with the Shell.) The CLI was a simpler version of the Shell.

Many of the functions the Shell gives you access to are also available from Workbench. The advantage of the Shell is that it gives you a finer degree of control over AmigaDOS functions. For example, let's say you want to list all the files in a directory whose names contain the letters "budget90." By typing the command

```
LIST #?budget90#?
```

in the Shell, you will get the list you want, but the Workbench offers no comparable way to do this. The Shell also lets you string commands together into scripts so that you can access multiple commands by entering a single command from the keyboard.

If you've worked with command-oriented computers such as MS-DOS or UNIX machines, you will have no trouble learning the Shell. On the other hand, if the Amiga is your first computer, you may find the Shell a little mysterious and intimidating. (You'll be in good company. After all, the people at Xerox PARC invented the graphical user interface to get away from things like the Shell.) Don't be scared away by the Shell; by following the examples in this book, you will get a good grasp of how the Shell works. Even if you're primarily a Workbench user, you'll find that it sometimes pays to dip into the Shell.

Programming with ARexx

The final Amiga OS interface covered in this book is the ARexx programming language. Some people may quibble with my characterization of a programming language as an interface, but I think ARexx more than fits the bill as an interface. Like the Shell's scripting function, ARexx lets you string AmigaDOS commands together to automate repetitive tasks. Because it is a language, however, ARexx gives you a greater degree of control over scripts than does the Shell, letting you introduce such aspects as arithmetic.

Although ARexx is an important interface to AmigaDOS, its most important function is to tie together different Amiga programs. ARexx lets you get inside programs and use their individual functions at will. For example, a single ARexx program can retrieve data from a disk using your database program, send the data to an associate over a modem using your telecommunications package, then format and print the data using your word processor.

Because it is a programming language, ARexx is far more complicated than Workbench or the Shell. The ARexx section of this book will give you a grounding in the language and provide a launching pad for your explorations.

Conclusion

Individually, the three interfaces to Amiga OS are quite powerful. Collectively, they give you unprecedented control over the operation of your Amiga. As you become proficient with Workbench, then the Shell, then ARexx, you'll discover that you are using each at different times for different tasks. No one interface is perfect for everyone, because no one interface can do it all. That's why the Amiga has three, and why you should become familiar with them all.

– 2 –
Workbench Basics

When you power up your computer you set in motion the *booting* procedure. Booting is computer lingo for the series of steps a computer takes to prepare itself for real work. It comes from the phrase "pick yourself up by your bootstraps" and can be used as a verb (boot up the machine) or a noun (perform a cold boot). Early personal computers, such as the Apple II, contained only enough software routines in permanent ROM (read-only memory) storage to read from the boot disk the code that loaded the disk operating system. Most of the Amiga OS software required by your machine resides in the 512K of ROM installed on the Amiga motherboard (main circuit board). The Amiga booting procedure's most important function is to configure the machine to your specifications.

Configuring the System

Turning on your Amiga sends a hardware reset signal to the CPU, which causes the machine to execute the boot code contained in ROM. This code consists of software routines that perform a number of diagnostic tests on your system and initiates the *Autoconfig* (short for auto-configuration) procedure. Autoconfig tells the system about any expansion devices attached to your machine.

Added to Amiga OS with version 1.2, Autoconfig makes living with your computer a lot easier. All Amiga expansion devices, whether they install internally or externally, use a standard set of signals to communicate with the Amiga. The Autoconfig portion of the standard lets an expansion device, such as a memory board, tell the system what the board is and how much space it occu-

pies in the system memory map. The system responds with the addresses it will use when accessing the device.

The best part of Autoconfig is that you don't have to understand the preceding paragraph to take advantage of it. Unlike MS-DOS expansion boards that require you to set all kinds of jumpers to keep them from stepping on each other's toes, Amiga expansion devices handle all that mumbo jumbo for you.

Disk Action

After the system configures all your expansion hardware, it searches for the boot disk. Originally, the Amiga could only boot from a floppy-disk drive. Version 1.3 of the Amiga OS changed that by letting you boot from a hard-disk drive whose interface card conformed to Commodore's autoboot standard. If you're using an Amiga 2000HD, 2500, or 3000, your system will boot from the internal hard drive. The same is true for an Amiga 2000, 1000, or 500 to which you've added an autobooting hard-disk system. On the other hand, if you're using a floppy-drive system or if you've deactivated the autobooting feature of your hard disk, you will have to boot from a floppy disk.

If you're booting from a floppy drive and you haven't yet put a Workbench 2.0 disk in the drive, the system will prompt you to do so when you turn on the machine. After you have inserted a bootable disk, the procedure is the same as for autobooting hard drives.

At this point, the startup code searches the boot disk for a file called *S:Startup-sequence.* This file is an AmigaDOS script that contains the names of the commands your system executes to configure itself. Script files in general, and Startup-sequence in particular, are covered in detail later in the book. For now, you need know only that one of the last commands in the standard Startup-sequence file supplied with every Amiga is LoadWB.

Commands listed in script files are actually programs, and LoadWB is the program that loads and manages the Workbench interface. (LoadWB is a misnomer. AmigaDOS loads LoadWB from the disk; the program does not load itself.) Workbench is simply another program running on your Amiga. While applications programs perform a specific function, Workbench makes controlling your system and running programs easier.

The Workbench Metaphor

If, like me, you had trouble distinguishing a simile from a metaphor in high-school English classes, you are probably not very happy seeing the word associated with your computer's interface. You can thank the folks at Apple for bringing the word into the vocabulary of personal computers. When it introduced the short-lived Lisa computer in 1983, Apple characterized Lisa's graphical interface as a metaphor for how people work at their desks. This "desktop" metaphor didn't catch on, however, until it showed up in the Macintosh.

This metaphor is a good characterization of a graphical user interface's function. Computers deal in intangible quantities; all information inside your computer consists of a series of +5 and +1 voltages. Thus, when you use a computer, you are doing nothing more than changing the state of millions of tiny electrical switches from +1 volts to +5 volts and vice versa. Software interprets these voltages as the 0 and 1 bits that make up all the instructions and data in a computer. The purpose of a graphical user interface is to relate the intangible aspects of computers to a concrete experience that everybody understands. The metaphor behind the Workbench, a program that exists only as 0s and 1s, is the workbench that you might have in your basement or garage.

What is a workbench? It's a place where you do work, store the tools to do the work, and keep the objects you're working on. The Workbench program (see Figure 2-1) turns your Amiga into a workbench where you perform work with tools and store the tools and objects you are using.

Tools, Drawers, and Projects

At a workbench, you get things done by manipulating tools. You hammer nails, drill holes, and saw wood. On the Workbench, your tools are programs; they let you manipulate data to get things done. For example, a word-processing tool lets you manipulate characters, words, paragraphs, even entire documents. A painting tool lets you manipulate the individual pixels (picture elements) that make up computer graphics.

Naturally enough, you store Workbench tools in drawers. These let you store similar tools together and label the drawer with a name that describes its contents. You can even store Workbench drawers within drawers, permitting you a great amount of latitude over how you organize your tools.

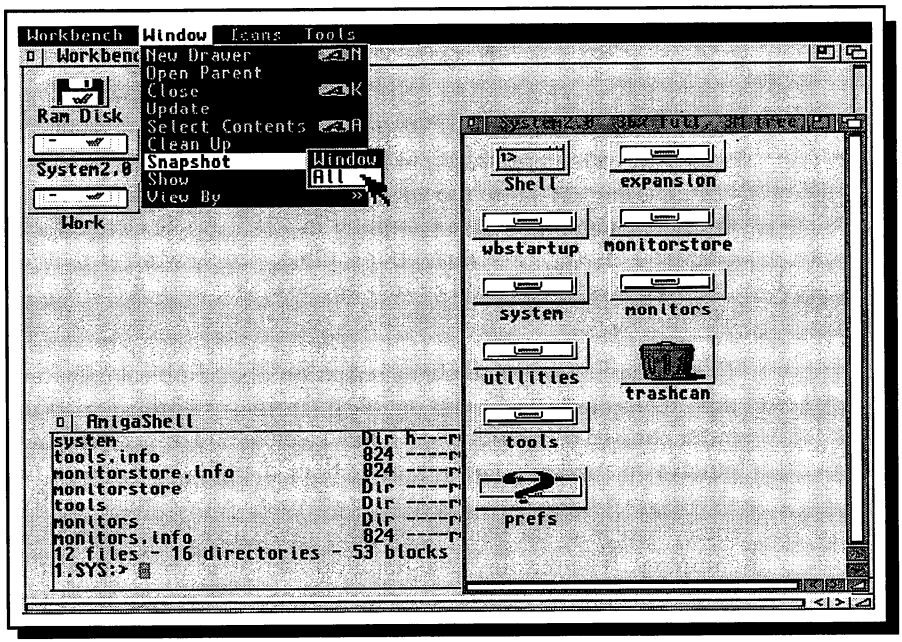


Figure 2-1 The Workbench Interface

Like all graphical user interfaces, Workbench combines windows, icons, menus, and a mouse pointer to let you control your computer. Unlike earlier versions, Workbench 2.0 features a sophisticated 3-D look.

In the real world, drawers exist in a chest. On the Amiga Workbench, you find drawers on disks. Drawers and their contents exist on your disks as files. As you'll see later, disks are a special form of drawers.

Finally, the objects that you manipulate with your tools are called projects. A project can be a letter created with a word processor or an animation created with an animation program. Projects created and manipulated with tools are stored in drawers. Thus, like tools, they are disk files. Another name for a project is a data file.

Files and Icons

In general, a file is a collection of related information (either program instructions or data) that resides on a disk. Files provide the permanence needed to make computers useful. In fact, your computer would be useless if it couldn't store things in a file, as you would lose all your work when you turned off the

power. Files give your computer persistent memory — memory that lasts after you turn your computer off. Every file has a name so that you or the Amiga OS can locate it. The job of the AmigaDOS subsystem is to store and retrieve disk files.

Because they are files on a disk, tools, drawers, and projects aren't something you can see and touch. To let you access them, the Workbench displays icons (small pictures) that represent these different objects.

Icons (see Figure 2-2) provide information about the the different objects on the Workbench. For example, the icon for a word-processing program might look like a typewriter or a pen. For practical purposes, you can consider an icon to be the object it represents. When you start up a program by double-clicking on its icon, you should think of that as manipulating the program directly, rather than simply manipulating a representation of the program.

Icons are actually files that contain the information about how they will appear on the Workbench display and about the tool, project, or drawer they represent. Icon files always have names that end with .info and are called "dot info" files. The first part of an icon's name is the file it represents. For example, Amiga OS 2.0's clock program is named Clock and the icon file is called Clock.info. Many Amiga files do not have corresponding .info files and thus don't normally appear on the Workbench display. You'll learn more about these files later.

Disks and Volumes

Disks and disk icons are special on the Amiga. When you first boot your system, the only icons you see are those that represent the disks active in the system. For a floppy-based system, these will usually be Workbench2.0 and Ram Disk. A hard-drive system may have more active disks, because you can divide the space on a hard disk into more than one partition, each of which acts as a separate disk.

Workbench calls disks volumes to differentiate between the disk and the drive that holds it. Thus, Workbench2.0 is the name of a particular volume, not the name of the drive that disk is in. When Workbench refers specifically to a disk drive, it uses the drive's AmigaDOS device name. The device name for the first internal drive in an Amiga is DF0: (for Disk Floppy 0). The second internal drive on an Amiga 2000 or 3000 is called DF1:, and the first external drive is DF2:. The first external drive on an Amiga 500 is DF1:. Workbench uses these device names instead of volume names for such functions as copying disks.

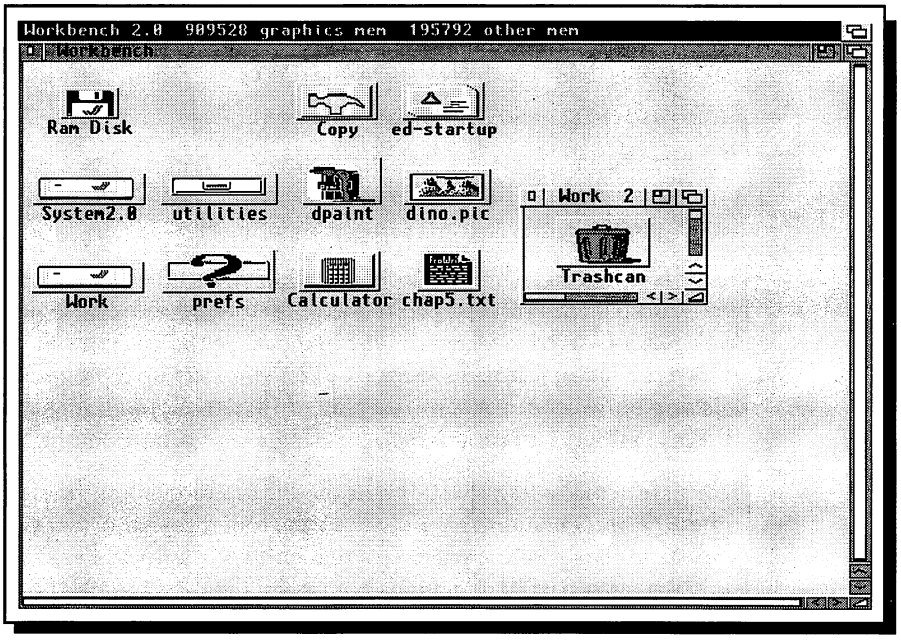


Figure 2-2 Workbench Icons

Workbench uses five types of icons: disk, drawer, tool, project, and garbage. At left are some disk icons. System2.0 is the version of the Workbench2.0 disk that Commodore installs on Amiga 3000 hard drives. Utilities and Prefs are drawer icons. The third column shows a group of tool icons. The top one is the default tool icon Workbench uses when you choose the Window Show All menu item. The fourth column consists of project icons. Again, the top one is the default used when you choose Window Show All. The last column shows the standard icon for the Trashcan. (Note: A trashcan must always be in a window.) You can create and modify icons using the IconEdit tool in the Tools drawer.

The Ram Disk is unique; as a piece of hardware, it doesn't exist at all! The Ram Disk is a section of memory that AmigaDOS reserves and treats like a disk drive. The advantage is that you can access files in the Ram Disk much faster than you can on a physical disk drive. The negatives are that a Ram Disk uses your precious memory and that the information in it is lost when you turn off your machine. Use it only as a temporary storage place.

Drawers and Windows

Workbench and its counterparts on other computers (Finder on the Mac, Windows 3.0 on MS-DOS machines, Presentation Manager on OS/2 comput-

ers, and so on) are often called windowing interfaces because they present information inside rectangular areas (windows) on your monitor display.

Windows regulate communication between you and the program that creates the window. A program sends its output to you via its windows and cannot send data to the video display outside the boundaries of its window or windows. Conversely, you can only input data to a program when you have selected its window using the mouse pointer — more on this in the next section.

On the Amiga, then, a window constitutes a "virtual" terminal. (An aside: I love the way the computer industry uses the word virtual to describe anything that isn't real. Imagine how many championship banners would be flying at Fenway Park if the Red Sox could label all their losses "virtual wins.") When you select the output window of a particular program, the keyboard and the window connect to create a virtual terminal patterned after the physical ones connected to big, time-sharing computers. You can thus have one terminal for each program running on your system, ensuring that you send input to the proper program and that each program keeps its output separate from that of the others.

Because Workbench is a program that manages your system and launches other programs, it uses windows in a unique way. The Workbench's output is the icons that represent the objects you can manipulate. As do other Amiga applications, it displays these icons in windows. The difference is that in Workbench you can manipulate icons to change the output of Workbench. You can move an icon from one window to another.

As I noted above, Workbench lets you organize tools and projects in drawers. Like other objects on the Workbench, drawers have icons you can manipulate. The unique property of drawers is that when you open one, Workbench displays a window that shows the objects in the drawer. Each drawer opens a window to display its contents, even if the contents include another drawer.

As an example, consider Figure 2-3, which shows the window that displays the contents of my drawer called Word Processing. Inside is my word processing program and three drawers: Book, Articles, and Letters. Each of these drawers contains other drawers. The Book drawer, for example, contains a drawer for every chapter of this book. By organizing my files and drawers in a logical manner, I can find files quickly and easily. This may seem trivial when you have only ten or twelve files and one floppy disk, but when you amass thousands of files on a hard disk the ability to find your files can't be taken too lightly. Drawers let you organize your disks and view their files in a way that makes sense.

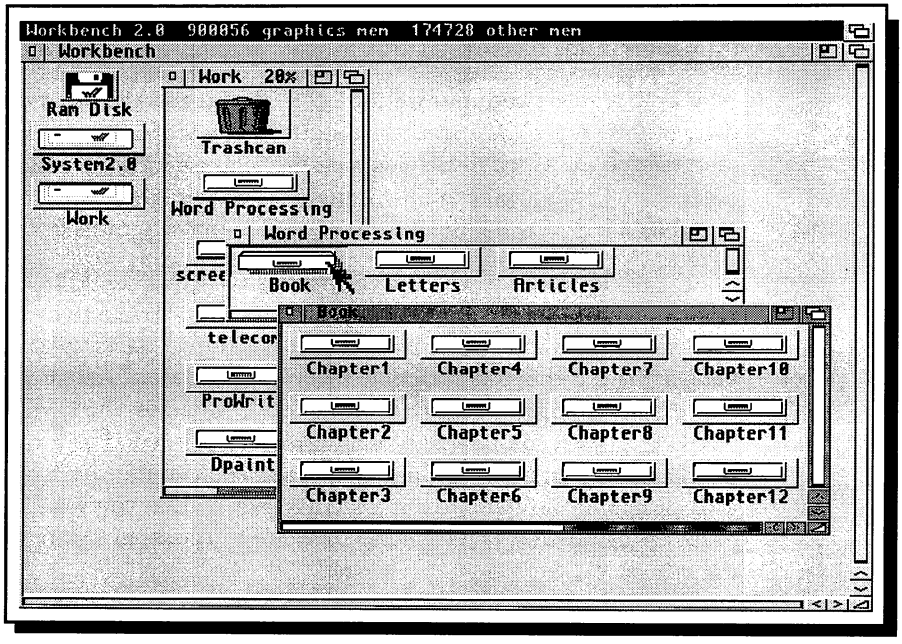


Figure 2-3 Organizing a Disk

Logically arranged drawers help you to find files quickly. I organize my word processing drawers by the types of files I store in each. I keep material for this book separate from material for articles, and keep both separate from correspondence. In the book drawer, I use other drawers to segregate material for the individual chapters.

Although disk icons look different from drawer icons, they work in a similar fashion. When you open one, Workbench displays a window that shows the contents of the disk, which may include one or more drawers. (A disk window, however, can never contain the icon of another disk.) The amount of free space on the disk is stated in its drag bar. The distinction between a disk window and a drawer window is small but important: If you drag an icon between two windows of drawers on the same disk, you are moving the object. If you drag an icon between the windows of two drawers on different disks, you are copying the object.

Windows on Workbench, therefore, have an important role: They display the contents of drawers and disks, providing the tangible objects you can see. To let you manipulate the objects, however, Workbench relies upon the mouse.

The Pointer and the Mouse

The Workbench metaphor goes only so far; on the Amiga's Workbench you can't simply reach out your hand and grab the icon for a spreadsheet program or a document as you can a hammer or board. To manipulate the objects you see, Workbench supplies a pointer that you control with your mouse.

When you move your mouse on the surface of a desk or table, the mouse mechanism sends signals about its relative movement to Intuition, the part of Amiga OS that manages the windowing interface. Intuition interprets these signals and moves the system pointer (also called the mouse pointer) on the display screen to correspond to the physical movement of the mouse. Thus, the mouse gives you a way to navigate the Workbench screen and point at icons. The mouse's two buttons let you act on what you see.

The Selection Button

Before you can work with an object on the Workbench, you have to select it; that is, indicate to the Workbench program you want to work with the object. This is the job of the left mouse button.

The left mouse button lets you select icons, gadgets, and windows. To select a project icon, for example, you first move the pointer over the icon and then click (press and release) the left mouse button once. The image of the icon will change to a complementary image to show that it is selected and active (see Figure 2-4). Note that when you select an icon, you automatically make inactive the icon on which you previously clicked. When you click outside of any icon on the Workbench, you make all selected icons inactive.

The same procedure applies to windows. If you have multiple programs running on your Workbench, you can select one by moving the pointer to anywhere within the window and clicking the left mouse button. Only one window can be selected at one time. The selected window then becomes the active window; its menus appear in the screen menu bar and it can take input from the keyboard. (The menu bar is the province of the right mouse button, which is discussed below.) You can tell when a window is selected because its title bar (containing the name of the window) will be blue and its scroll bars, if present, will be white (see Figure 2-5). Inactive windows are all grey. When you first boot up, the Workbench window is the only one open (although you will learn how to change that) so it will naturally be the one selected.

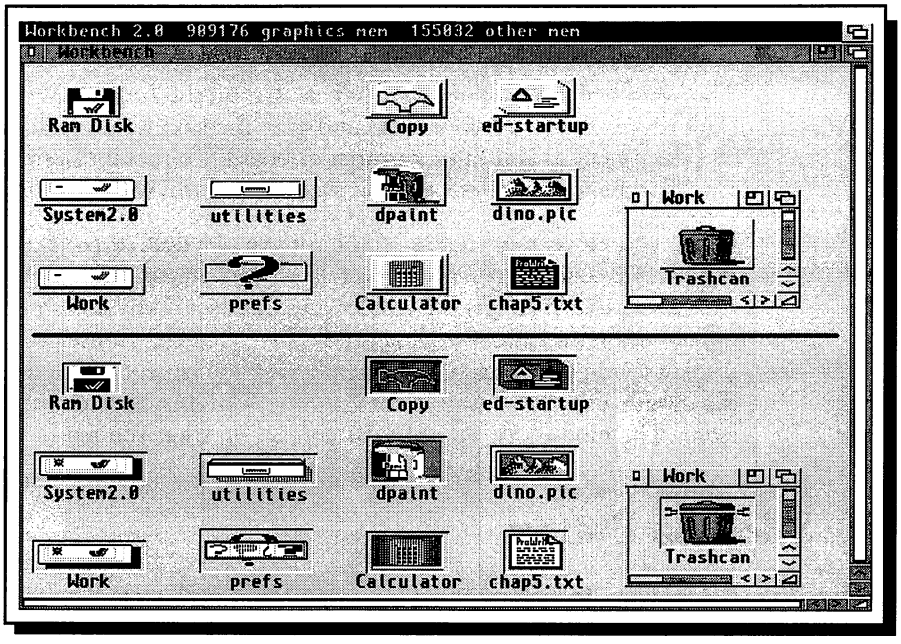


Figure 2-4 Selected Icons

The top group of icons are unselected. The bottom group are the same icons after they have been selected and are active. Unselected icons on the Workbench appear to be above the plane of the display, and selected icons below the plane. Commodore creates this effect by the use of white and black lines. Unselected icons have white borders on their left sides and tops; black borders on their right sides and bottoms. Selected icons use opposite imagery. Some icons display alternate imagery when they are selected.

Icons are not the only object that you can select with the left mouse button. You also use it to select gadgets. For example, if you've opened a drawer into a window, you can close the window by clicking on the close gadget in the upper-left corner of the window. More on gadgets later.

The Double-Click

While a single-click on an icon selects that object, a double-click — clicking twice in rapid succession — opens the object. Opening an object has different effects depending upon the type of object. With a disk or a drawer icon, for example, double-clicking opens the corresponding window, which displays the contents of the disk or drawer (see Figure 2-6). So you can see for yourself, let's open your boot disk.

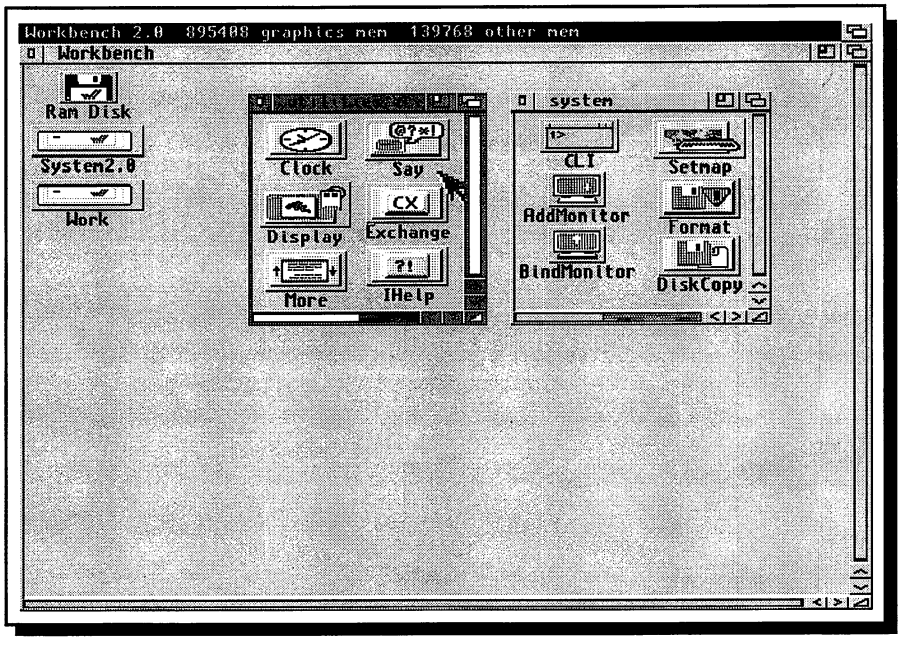


Figure 2-5 Selected Window

You can tell the currently selected window on the Workbench by the condition of its title bar (drag bar) and scroll gadgets. On unselected windows, the title bar and scroll gadgets are grey. On the selected window, however, the title bar is blue and the sliders in the scroll gadgets are white. The selected (active) window is the only one that receives input from the keyboard and mouse.

The display you get when you boot your system depends upon whether you booted from a floppy drive or a hard-disk drive. If you booted from a floppy and had no other disks in the drives, you will see two icons on your screen — one labeled Ram Disk, the other Workbench2.0. If you booted from a hard drive, again with all other drives empty, you will see three icons: Ram Disk, Work, and System2.0. Depending on your system, open either the Workbench2.0 or the System2.0 disk by double-clicking on its icon. You should see a display similar to Figure 2-7. The window will open to show the disk's name in the upper left and the icons that represent the contents of the disk.

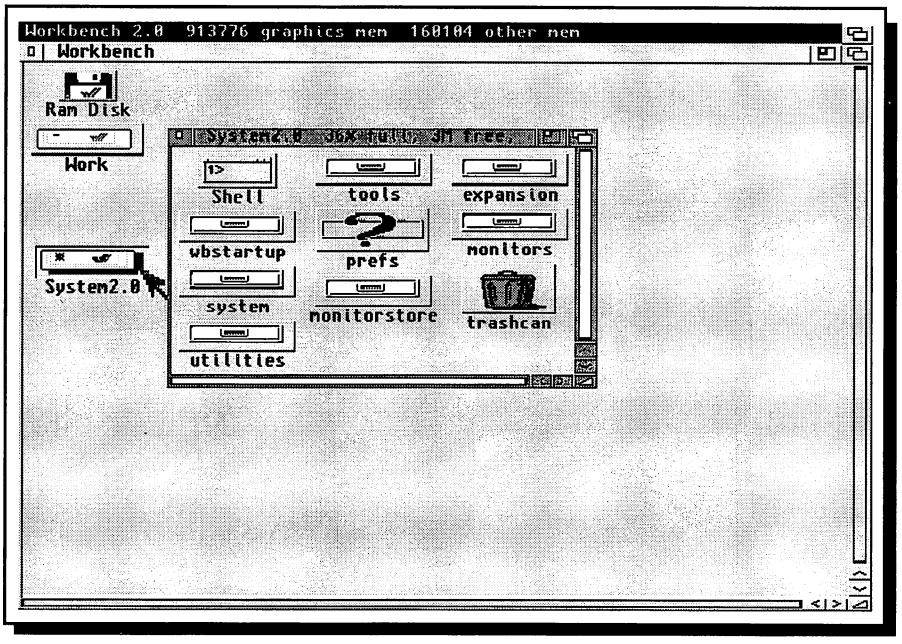


Figure 2-6 Icons and Windows

The relationship between icons and windows is straightforward: Opening a disk or drawer icons causes Workbench to display a window showing the contents of the icon. An icon and its window have the same name, as shown here by the System2.0 icon and window.

You can open tools in two ways. If you open a tool icon by double-clicking, Workbench will load and run the corresponding program. If you open a project icon, Workbench will find the tool that created the project, load the tool, and then load the project into the tool. Thus, you can load and run your word processor by simply double-clicking on the icon of any data file created by the word-processing program.

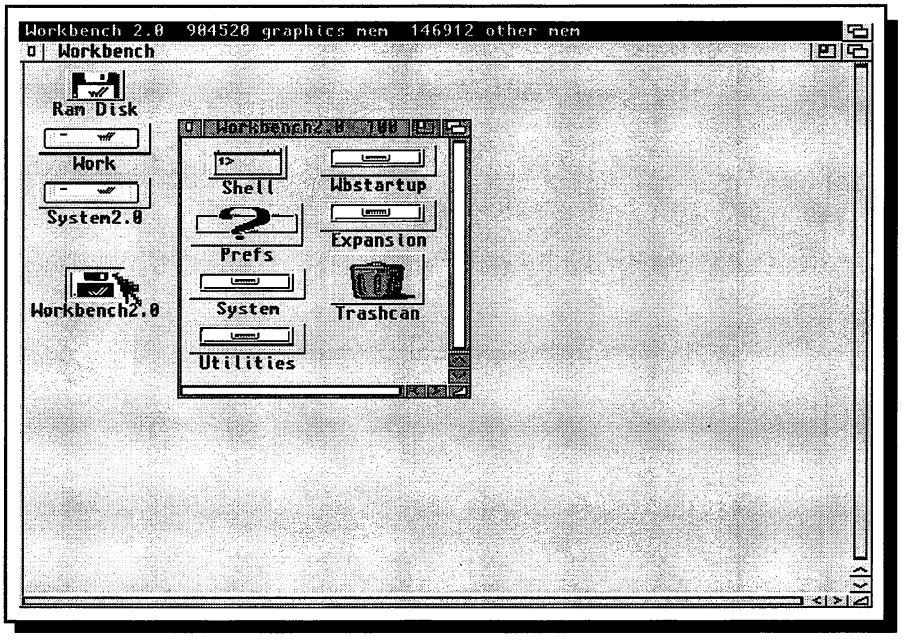


Figure 2-7 Workbench2.0 Window

The primary boot disk for floppy-disk-based machines is Workbench2.0. When opened, it reveals one tool, the Shell, five drawers, and a trashcan. If your Amiga came with a hard drive already installed, your boot disk is probably called System2.0 and may contain some drawers normally found on the Amiga Extras disk.

Dragging Icons

The left mouse button can perform other actions on icons if you press the button down and *hold it.* This both selects the icon and "glues" it to the pointer. For example, to copy a document to a second disk for safe keeping, you simply press and hold the left mouse button with the pointer on the file's icon and move the pointer, with the attached icon, over the icon of the second drive. Once it is positioned over the disk icon, you release the button, and Workbench copies the file and its icon to the second disk (see Figure 2-8), leaving the original file on the first disk.

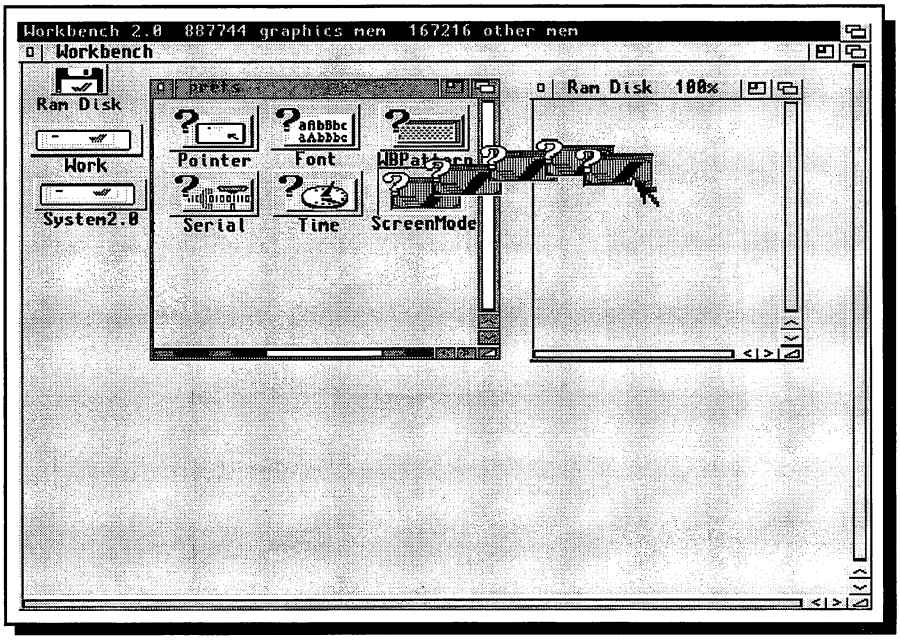


Figure 2-8 Copying a File

To copy a file using the select button, you simply move the file's icon from the drawer of one disk to the drawer of another. First, click on the icon and hold down the select button. Next, drag the icon to the new drawer. Finally, release the button and Workbench will place a copy of the file in the new drawer.

If you have only one disk drive, copying a file to a second disk is more involved. The easiest method is to copy the file from the original disk to the Ram Disk. Then, eject the first disk from the drive and insert the second disk. (Warning: *Never* eject a disk while the disk-drive light is on. Always wait a few seconds after the light goes off before pushing the disk-eject button.) Finally, open the Ram Disk, grab the icon you want to copy (by pressing and holding), and move it to the second disk, releasing the button when the pointer is positioned over the second disk. Don't worry about that extra copy in the Ram Disk, you can delete it immediately if you need the room; otherwise, it will disappear when you turn your machine off.

In addition to dragging tool and project icons, you can also drag drawer and disk icons. Like dragging tools and projects, dragging a drawer icon to another drawer or window on the same disk moves the drawer and its contents. Dragging it to the disk icon of another disk or to a drawer or window of another disk, copies the drawer and all its contents to the second disk.

You can also copy entire disks by dragging disk icons. If you drag the icon for a floppy disk to the icon of another floppy disk, you will copy the contents of the first disk to the second, erasing the latter's contents. If you drag any disk icon to the icon of a hard-disk partition, Workbench creates a drawer in the hard-disk partition with the same name as the source disk and then copies as much of the contents of the source disk into the new drawer as will fit.

Extended Selections

Although you can have only one active window at a time, you can have more than one icon selected at a time. If you wanted to move several icons from the System drawer to the Ram Disk, for example, you would not have to select and move them individually. You could select them all and then move them as a group, simplifying the process of copying a series of related files. Workbench also offers menu functions that work on multiple icons simultaneously. You can select more than one icon with the selection button via two procedures.

Once you've selected one icon, you can select another without deactivating the first by holding down either of the Shift keys while you click on subsequent icons. To see how this works, open the System drawer on your boot disk and select the DiskCopy icon. Now, hold down one of the Shift keys and single-click on both the NoFastMem and SetMap icons. When you are through, you'll see three selected icons.

New to version 2.0, the second method of activating more than one icon is called *drag selecting.* To demonstrate, press the selection button while the pointer is inside the System window but not over any icon. Hold the button down and move the pointer. Note the rectangle that follows the pointer. Move the rectangle so that it encompasses a number of icons. When you release the button, all the icons that fall partially or completely within the box are selected. Note that even with many selected icons, the next time you click on an icon, all the others will become inactive unless you use the Shift key.

The Menu Button

A graphical user interface not only lets you select and open objects easily, but it also gives you the ability to send commands to programs without great effort. The right mouse button gives you this capability. It lets you control your Amiga's pull-down menus, and is thus called the menu button.

Every program running on the Amiga can create its own menu system (although not all do). Workbench is no exception. To see the menus provided by Workbench, select (with the left mouse button) the main Workbench window, the one where the disk icons reside. At the top of your display, you'll see a black bar that reads "Workbench 2.0 xxxxxx graphics mem xxxxxx other mem." Now, press and hold the menu button. The bar now changes to reveal the four Workbench menus — Workbench, Window, Icons, and Tools (see Figure 2-9).

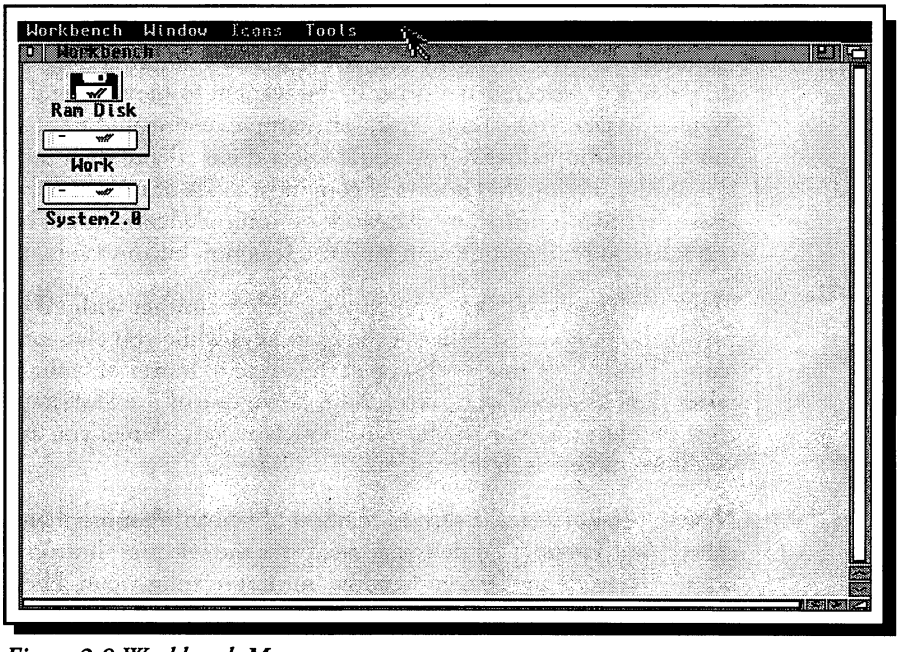


Figure 2-9 Workbench Menus

You find the Workbench menus in the title bar of the Workbench screen. They are visible when you hold down the menu button on the mouse. The Icons menu remains ghosted until you select an icon.

If, while still holding the menu button, you move the mouse pointer to the Workbench menu, the items in the menu will appear (see Figure 2-10). As you can see, the items in the Workbench menu are Backdrop, Execute Command, Redraw All, Update All, Last Error, Version, and Quit. As you move the pointer over each of the other menus, their contents will drop down in turn.

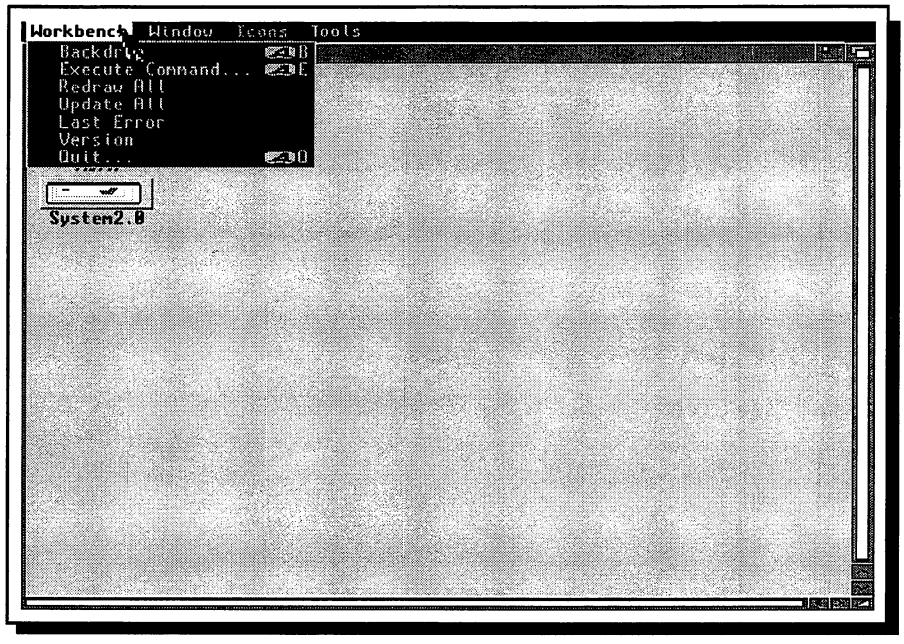


Figure 2-10 The Workbench Menu

To display a menu's items, hold the menu button down and move the pointer to the menu. Items in the Workbench menu affect the Workbench environment as a whole.

Accessing a menu item is easy. Once again, hold the menu button and pull down the Workbench menu. Now, while still holding the button, move the pointer down to the first item, Backdrop. The Backdrop item will be highlighted — it will appear blue on white. To perform the Backdrop function, or any menu item function, you simply release the menu button while the item is highlighted (see Figure 2-11). So, release the button (your finger must be getting tired).

What happened? At first glance the Workbench window seems to be gone. Your disk icons are no longer inside a window, and they aren't surrounded by a border. They are, in fact, still in the Workbench window, but the nature of the window has changed. It has become a backdrop window, which, by definition, doesn't have a title bar, borders, or any of the other normal window attributes. Also, a backdrop window is always behind all other windows on the screen. In fact, with previous versions of Workbench (1.3 and older), the Workbench window was always a backdrop window. It's only with Workbench 2.0 that you can manipulate the main Workbench window as you can other windows.

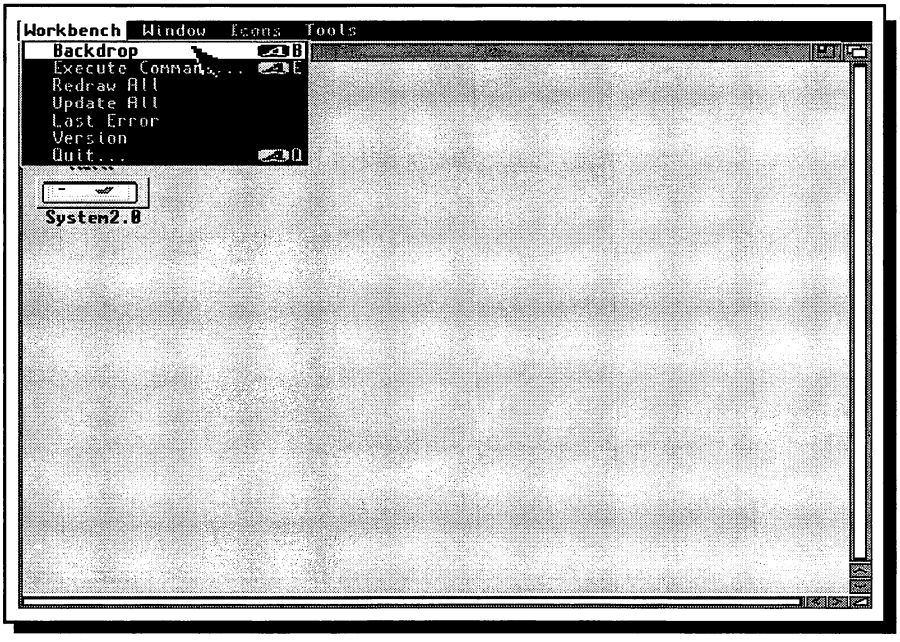


Figure 2-11 The Backdrop Menu Item

To choose a menu item, move the pointer over it while holding down the menu button. The item — the Backdrop item, in this case — will show up as blue on white. By releasing the button you execute the Backdrop function. Note the keyboard equivalents for Backdrop, Execute Command, and Quit.

If you don't like the backdrop Workbench window, hold the menu button and pull down the Workbench menu again. This time, the Backdrop item has a check next to it, meaning that you've turned it on. Now, move the pointer to the Backdrop item again and release the button. Your Workbench window is back!

Backdrop is an example of a toggle item. It works like a toggle switch, such as an ordinary light switch. Press it once, and it turns on; press it again, and it turns off. When you first choose Backdrop, you are turning the backdrop option on; choosing Backdrop again turns the option off. Note that Backdrop only works on the main Workbench window, not on any drawer or disk windows.

The Workbench Screen

Unlike the other objects discussed, the black bar that holds the Workbench menus is not connected to a window. It's a part of the Workbench screen.

On the Amiga, a window is a region that displays a program's output. A screen is a region that displays windows and defines the environment a window appears in by its colors and pixel resolution. A screen also makes menus available to its subordinate windows. Any window that a program opens will inherit the colors and resolution of the host screen and use that screen's menu bar. This lets you run programs that require different resolutions on your Amiga. For example, if your word processor needs to run on a screen that is 640 pixels wide, and you want to run your paint program at 320 pixels to get the maximum number of colors, you can run the programs at the same time on different screens. Any program that requires a number of colors, resolution, or display mode — such as hold-and-modify (HAM) — that isn't supported by the Workbench screen can open its own screen. Unless you are a programmer, the only thing you have to know about screens is that they control access to menus.

Try an experiment that demonstrates this relationship: Open your Workbench2.0 or System2.0 disk. From the window, open the Utilities drawer. In the Utilities drawer, open the Clock icon. The Clock icon starts the Clock tool, which produces the Clock window. Move your pointer to the Clock window and select it by clicking anywhere within its borders. Now, hold down the menu button. Notice that the menu bar no longer carries the Workbench menus, but now contains the menus for the Clock program. When you press the menu bar, it will always show the menus for the currently selected window. To get back the Workbench menus, you need only click inside the main Workbench window or one of its subordinate windows.

Remember, no matter where a window is on your display, its menu bar, which is active only when the window is selected, will always be found on the screen title bar.

Windows In-Depth

As you can tell by looking at your display, there is more to a window than a rectangular array of pixels. Around the edge of most windows are a number of items called gadgets. These permit you to perform standard functions on windows.

To see how gadgets operate, look at a sample window. Open your Workbench2.0 (or System 2.0) disk, and then the Utilities drawer. Like all windows created by Workbench, this one has seven gadgets. Clockwise from the top-left, they are the close gadget, drag gadget (the title bar), zoom gadget, depth gadget, vertical scroll gadget, sizing gadget, and horizontal scroll gadget (see Figure 2-12). These let you control various aspects of a window to give you more control over your Workbench environment.

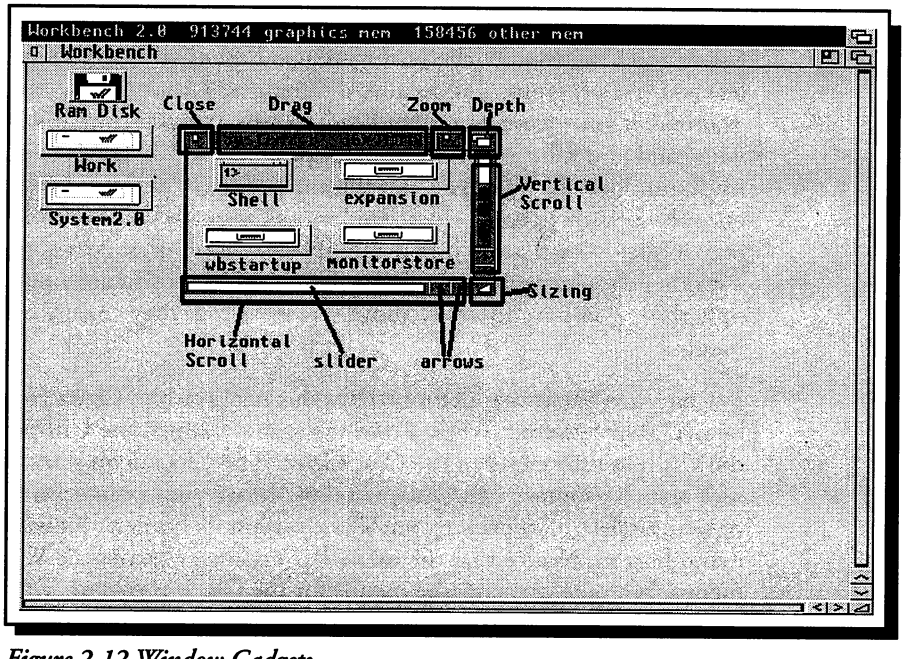


Figure 2-12 Window Gadgets

The standard window gadgets are close, drag, zoom, depth, vertical scroll, sizing, and horizontal scroll. The scroll gadgets are a combination of a proportional gadget and two button gadgets.

Close Gadget: As its name implies, the close gadget closes a window. If the window is associated with a disk or a drawer icon, the window will simply disappear from your display. If another program created the window, the close gadget will usually shut down the program and remove the window. If you click on the close gadget on the main Workbench window, however, the system displays a message — called a requester, because it is requesting input — asking you if you are sure you want to shut down the Workbench window. In most circumstances, your answer should be no. Closing Workbench now would leave you no way to control your computer! When you learn how to use the Shell, you will be able to close Workbench safely. To see the close gadget

in action, select the one on the Utilities drawer. It removes the drawer from the Workbench.

Drag Gadget: The drag gadget is the thick horizontal bar at the top of a window that doubles as the window's title bar. (Note that the name of its icon is the same as the name of its window.) The drag bar lets you move a window around on its screen. To do so, first, reopen the Utilities drawer, then position the pointer anywhere on the title bar. Next, press and hold the select button. Now, move the pointer. Note that the window moves as if hooked to the pointer. This is called dragging the window. In general, pressing and holding the select button on an object, and then moving the object, is called dragging.

The drag bar of a disk window and a drawer window differ in one respect only. A disk window has more information in its drag bar than the window name. For example, the Workbench disk window on my Amiga 2000 says "Workbench2.0 97% full, 29K free, 850K in use." The figures tell you the amount of space you have left on the disk to store files. An abbreviation for kilobyte, K is a measure of computer storage that corresponds roughly to 1000 bytes. Each byte can hold a single character, such as the letter N.

Note that because the Amiga allows for multiple screens at the same time, screens also can have drag gadgets in their title bars so that you move a screen to see the one behind it.

Zoom Gadget: On the right side of the drag bar, you'll find two gadgets. The left-most one is the zoom gadget.

Commodore's documentation on the zoom gadget is among the most non-committal writing I've read. According to the manual, the zoom gadget "changes the size of a window." Well, not necessarily. It also states that "the zoom gadget on the Workbench2.0 disk window expands the window to the full width and height of the screen." This is only true part of the time as well.

To use a zoom gadget correctly, you have to remember that each one has a memory. More precisely, each one has access to the information the system maintains about its window. It uses this information to decide what to do when you select the zoom gadget.

To see what I mean, select the zoom gadget on the Utilities window. The window shrinks to its minimum size. Now, select the gadget again. The window opens to its original size. The zoom gadget, therefore, is similar to a toggle switch.

The confusion comes from the fact that you can set how big or small the window becomes *and* where the window appears when you hit the zoom gadget. For example, with the Utilities window at its original size, drag it as far as you

can to the bottom left of your display. Now, select the zoom gadget. The window moves to its original position without changing size. Select the zoom gadget again, and the window returns to its new position at the bottom left of the display. What is going on?

The process is not very mysterious. Every window keeps information on its previous position and dimensions in addition to its current position and dimensions. When you select the zoom gadget, you're telling the window to move to its last position and resize itself to its last dimensions. There need be no correlation between these two positions. The Utilities window shrank to its minimum size the first time you clicked on the zoom gadget only because that is the default size Workbench presets. You can change the presets with the drag bar and the sizing gadget.

Depth Gadget: To the right of the zoom gadget, in a window's upper-right corner, is the depth gadget. Selecting the depth gadget not only makes a window active, but it also affects the window's placement on the Workbench screen.

When opened, each window on Workbench is assigned a unique depth that tells Workbench which windows or parts of windows to hide and which parts to display when two or more windows overlap. Windows are assigned a depth in the reverse order of their opening, with newly opened windows on top of older windows. Even when two windows don't overlap, they are still considered to be at different depths on the Workbench screen, just in case they do overlap later.

If you click on the depth gadget for the Workbench screen's top-most window, you send that window to the back. If you hit the depth gadget on any other window on Workbench, including the bottom-most one, you bring that window to the front. The depth gadget lets you decide which windows to display up front on your display and which ones to relegate to the rear.

The Workbench screen also has a depth gadget. In case one or more programs open their own custom screens, this gadget lets you flip between screens. If a screen does not have a depth gadget or a drag gadget, however, you can still get back to the Workbench screen by pressing the Left-Amiga key and the N key at the same time. This brings the Workbench screen to the front of all screens open on the system. Pressing Left-Amiga-M moves the Workbench screen behind all screens on the system.

Sizing Gadget: In the lower-right corner of a window sits the sizing gadget, which lets you enlarge or shrink the window. To see how it works, once again open the Utilities window. With the drag bar, move the window to the top-left of the screen. Now, move the pointer to the sizing gadget and press and hold the select button. Without releasing the button, move the pointer to the lower-

right corner of the screen. Release the button. You've just made the Utilities window fill the entire screen. In addition to making windows larger or smaller, the sizing gadget can change their shape. For example, by dragging the sizing gadget around, you can make your windows tall and narrow or short and squat.

Horizontal and Vertical Scroll Gadgets: Located at the bottom and right of a window, respectively, the horizontal and vertical scroll gadgets each consist of a slider and two arrow gadgets. Often, a window has too many icons to be drawn inside its bounds. To see the icons not immediately visible, you can either use the sizing gadget to open the window to its maximum size or you can scroll around the window with the scroll gadgets. Note that the horizontal scroll gadget only concerns itself with hidden icons to the left and right of the window's visible area. The vertical scroll gadget is concerned with icons above and below the visible confines of the window. Let's try scrolling.

Open the Prefs drawer on your Workbench or System disk and resize the window so that only the Input, IControl, Printer, and Overscan icons show. Note that the white slider, which previously filled its box entirely, now fills only a portion of it. The size of the slider in relation to its box is in the same proportion as the size of the window's visible area in relation to the window's total size (see Figure 2-13). Thus, a window large enough to display all its icons would have two sliders that completely fill their boxes.

The sliders not only indicate the percentage of visible icons, but also their relative location. If the horizontal slider is all the way to the left, you can find additional icons by moving the window display to the right. If the slider is in the middle of the box, you have undisplayed icons to both the left and the right of the current view.

Once you know where the icons are, you can use both the slider and the arrows to move the window display to see them. You can grab the slider (by pointing to it, then pressing and holding the selection button) and drag it to the left or right (or up or down) by moving the pointer. When you release the mouse, you'll see the icons in the area indicated by the new position of the slider. You can also move the slider by clicking inside its box (but outside the slider) with the selection button. Each click moves the slider one "view" to the left or right (or up or down), depending on whether you click to the left or right of (or above or below) the slider. The third way to scroll through a window is by using the arrow gadgets. Selecting the right arrow gadget scrolls the window a small amount to the right. The left arrow gadget moves it a small amount to the left. Pressing and holding either of these gadgets will cause the window to scroll continuously. Try these three methods with the reduced Prefs window.

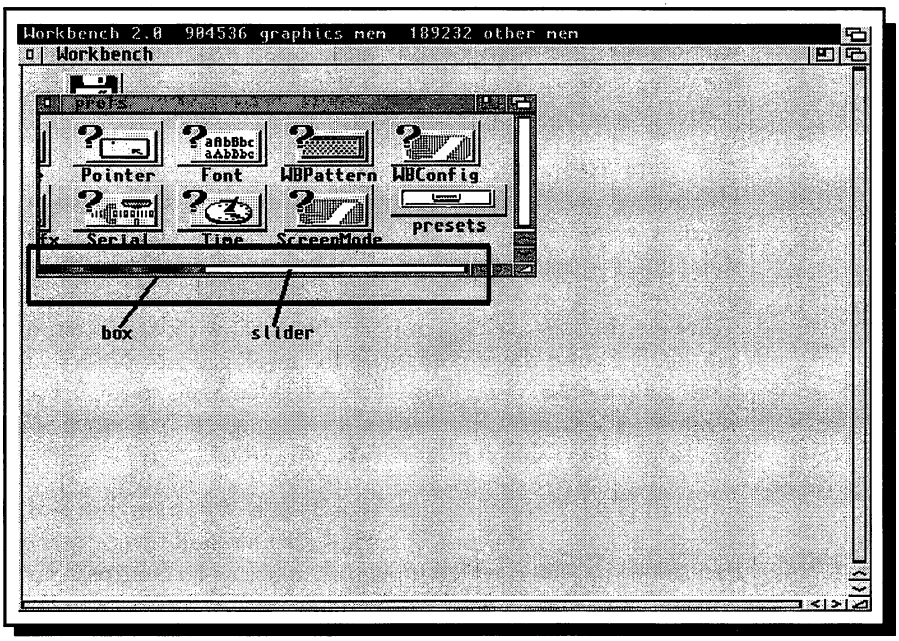


Figure 2-13 Proportional Gadget

The size of the slider in a scroll gadget tells you the proportion of a window's icons that are visible. When all the icons are visible, the slider fills its box. In this example, the slider indicates that there are more icons to the left of the current window view.

Gadgets and Requesters

Gadgets appear in many places besides Workbench windows and in more varieties. In fact, Commodore supplies a large number of standard gadgets that Amiga programmers can customize and use in their own applications. Often, Workbench or an applications program puts one or more gadgets into requesters (small windows that ask you questions). You normally have to satisfy the program's request for information before you can proceed. The simplest requesters ask you a yes/no question such as "Do you really want to delete that file?" You respond by typing an answer or by clicking on a gadget. By understanding the basic gadgets used in requesters, you should be able to respond to any requester you come across. The basic requester gadgets are buttons, sliders, and strings.

Buttons: Most of the Workbench window gadgets are variations on a button gadget. These gadgets expect you to use the select button to make a choice. For example, when you use the Format item in the Workbench Disk menu to prepare a blank disk to hold data, a requester appears asking if you are sure you wish to format the indicated disk. (Formatting a disk is a big step; it will erase the previous contents of the disk.) The requester displays three buttons; you indicate your choice by simply selecting one (see Figure 2-14).

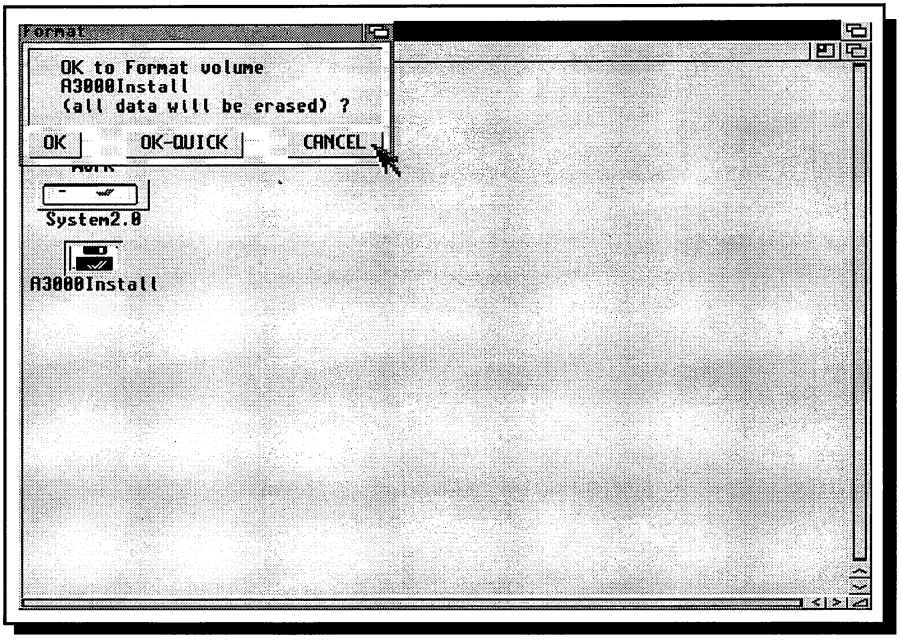


Figure 2-14 Simple Requester

When you select the Format Disk option from the Icons menu, the system asks if you are sure you want to format the indicated disk. Selecting the Continue gadget tells Workbench to proceed with formatting. Selecting Cancel tells Workbench not to format the disk. If the disk was previously formatted, a third button, OK-Quick, tells Workbench to use the old format of the disk but to consider the disk empty. (In technical terms, the OK-Quick option rewrites only the root blocks of the disk.)

Sliders: You've already seen one type of slider — a proportional one, in the discussion of the vertical and horizontal scroll gadgets. Most sliders are simpler than those and merely let you choose from a range of numeric values.

The most common example of a requester that uses a slider is a color requester. The Amiga builds colors using four bits of information on the red component

of a color, four bits on the blue component, and four bits on the green. A color requester, like the one shown in Figure 2-15, has three sliders, one for each color component. You use the sliders to select one of sixteen positions — representing the sixteen values, 0 to 15, that you can express with four bits of information (2 raised to the fourth power equals 16). You manipulate sliders by either dragging the slider image itself to the desired value or by clicking in the "track" that the slider moves in. Clicking in the track moves the slider towards the pointer.

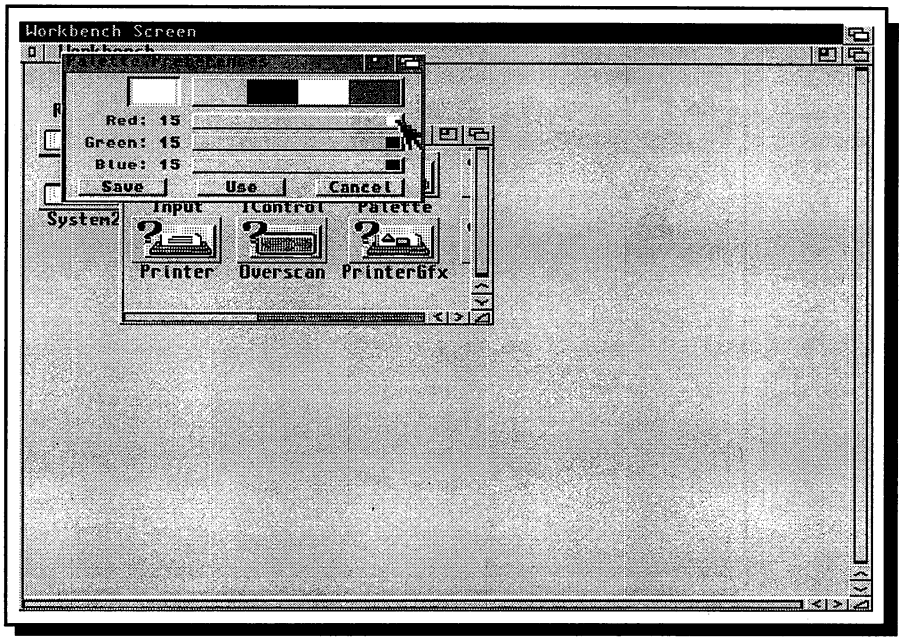


Figure 2-15 Requester With Sliders

When you open the Palette tool from the Prefs drawer, the Palette Preferences requester appears. By dragging one of the sliders or clicking in the track to the left or right of a slider, you move it to a position from 0 to 15. You select the color you want to change by clicking on it. The box at top-left shows the color you are altering. The Save button saves the changes to disk for immediate use and for use when you next boot your system. Use lets you work with the new colors but does not save them, and Cancel reverts to the original colors.

String gadgets: The most versatile gadgets are ones that put the fewest restraints upon the information they let you enter. String gadgets, also called text gadgets, are the best examples of these. A string gadget lets you enter information from the keyboard. For example, when you rename a file using the Re-

name option from the Workbench Icon menu, a requester containing a string gadget appears and lets you enter the new name of the file (see Figure 2-16).

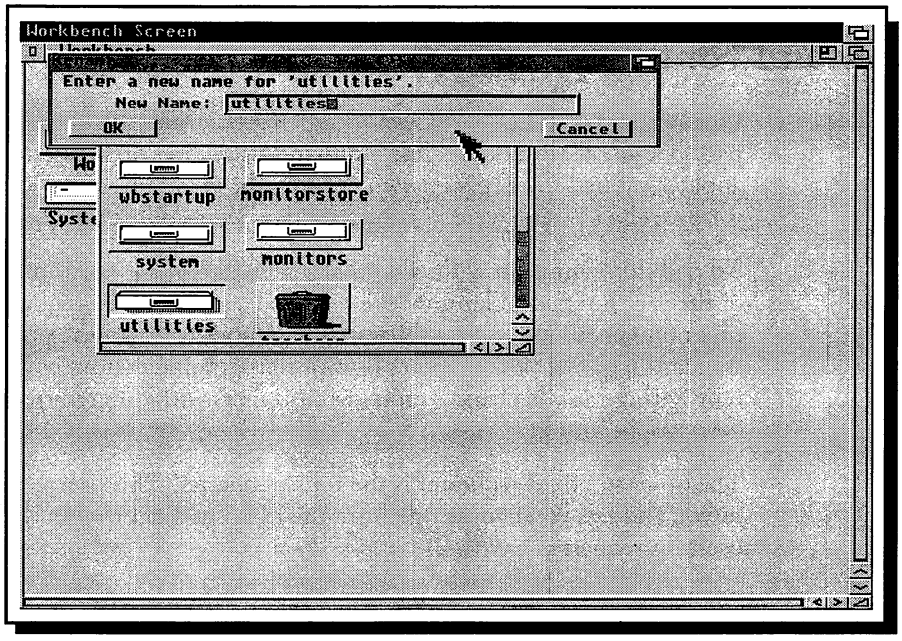


Figure 2-16 Text Gadget in a Requester

Choosing the Rename item from the Icons menu brings up the Rename requester. Use the keyboard to change the icon's name. Backspacing deletes the letter to the left of the cursor, while the arrow keys let you move the cursor without deleting characters. Typing a letter inserts that letter at the cursor position and moves the cursor and the following letters one position to the right. Selecting the OK button accepts the new name; Cancel reverts to the original name.

String gadgets all have a cursor — a rectangle that indicates where the next typed character will be placed. If you make a typing mistake in a string gadget, you can edit your input before you hit the Return key, which sends the information to the program that requested it. The Del key deletes the character under the cursor, while the Backspace key deletes the character to the left of the cursor. Pressing the Right-Amiga key and X simultaneously deletes the contents of the string gadget. Right-Amiga-Q restores what you deleted with Right-Amiga-X.

Many requesters combine more than one gadget. For example, the requester used with the Workbench Rename function also has a button that lets you

cancel the operation. While most requesters are customized for a certain operation, Commodore has provided two standard requesters with Amiga OS 2.0.

File and Font Requesters

One of the more frequently seen Amiga requesters is a file requester. In the past, because Commodore didn't provide a standard file requester, each Amiga application had to supply its own. The results were mixed: Some requesters were excellent, some were horrible, and few were consistent across applications.

To end this confusion with what is arguably one of the most important parts of the user interface, Commodore included a standard file requester in 2.0 for programmers to use in new software. Commodore hopes the presence of a standard requester will keep applications programmers from constantly reinventing the wheel and supplying customers with different requesters with every application.

The other standard requester is the fonts requester. This lists the fonts available to you and lets you choose one from the list. The Font tool in the Prefs drawer uses this requester.

Conclusion

This chapter introduced the major elements of the Workbench interface — icons, windows, menus, gadgets — and showed you how to manipulate them. Subsequent chapters build upon the information presented here.

Don't be put off if you still feel a bit uncomfortable with Workbench; you'll grow more confident as you use it. With a bit of practice, it will begin to seem like second nature.

– 3 –
Workbench At Work

The amazing thing about the Workbench system is that you can do so much with a simple two-button mouse. The trick to getting up to speed quickly with Workbench is understanding that the mouse buttons are complementary. The menu button lets you choose what to do, and the selection button lets you indicate what to do it to.

Chapter 2 focused mainly on the selection button. It is your primary manipulation tool, so you should master it first. This chapter introduces you to more of the functions of Workbench and its powerful menus. You already know how to do things; now you'll learn what to do and why.

Menus on Workbench

Workbench divides its menu items by functionality. The first menu — Workbench — contains items that affect the Workbench interface as a whole. Similarly, the Window menu contains items that work on the currently selected window, and the Icons menu items work on icons. The fourth menu — Tools — lets you access applications from Workbench. Of course, only applications that support this feature can be attached to and accessed from this menu. Because Workbench 2.0 is new, few applications currently take advantage of this feature. The only function provided with Workbench that uses the Tools menu is ResetWB.

Boot your Amiga and examine the Workbench menus. The Icons menu looks funny, doesn't it? If you pull down the Icons menu, you'll notice its items have the same appearance (see Figure 3-1). This look is called "ghosting."

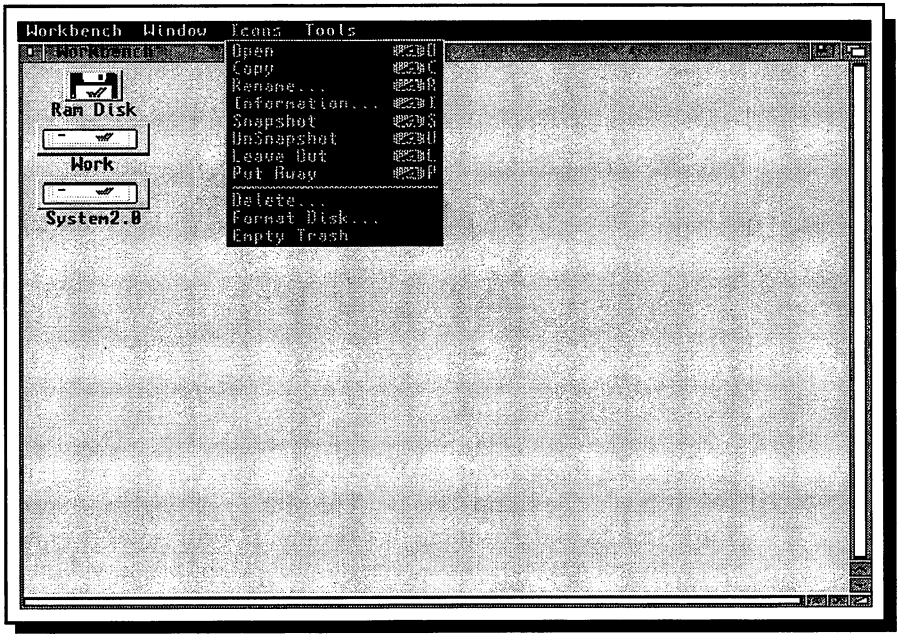


Figure 3-1 Ghosted Menu and Items

Workbench menus are context sensitive; when it is inappropriate or impossible to perform certain functions, Workbench "ghosts" those menu choices. The Icons menu, which only acts on icons you select, remains ghosted and inaccessible until you select an icon.

Menus and items are ghosted when they are inactive and cannot be accessed. Workbench and other Amiga applications use ghosting to keep you from performing operations that are inappropriate or downright impossible at a given time. Menus are context sensitive; they only let you do things that are possible at the moment. The Icons menu is ghosted when no icon is selected on the Workbench, because the items in this menu act only on icons. The Workbench and Window menus, on the other hand, are always active because Workbench is always active when it's open and always contains one active window.

The Workbench Menu

The seven items in the Workbench menu (Backdrop, Execute Command, Redraw All, Update All, Last Error, Version, and Quit) act globally upon the Workbench interface. They are always available when the Workbench is active.

Backdrop: As you learned in the previous chapter, the Backdrop item toggles the Workbench window between a standard window and a backdrop window. I did not point out, however, the meaning of the letters to the right of the item. The first letter, the A encased in a rounded rectangle, refers to the Right-Amiga key on your keyboard. The second letter, B, refers to the B key. Together, they form the "keyboard equivalent" of the Backdrop menu item. Pressing them simultaneously accesses the Backdrop item just as if you had pulled down the Workbench menu and chosen Backdrop. Many, but not all menu items have keyboard equivalents. On the Workbench, they always use the Right-Amiga key in combination with some other key. The Right-Amiga key, therefore, is sometimes called the menu key.

Execute Command: Like the Backdrop item, this one is new to Workbench 2.0. Choosing the Execute Command menu item, or typing its keyboard equivalent (Right-Amiga-E), brings up a requester with a string gadget (see Figure 3-2). The requester prompts you to enter a command and its arguments. Two buttons give you the option of executing the command you enter (OK) or forgetting the whole thing (Cancel).

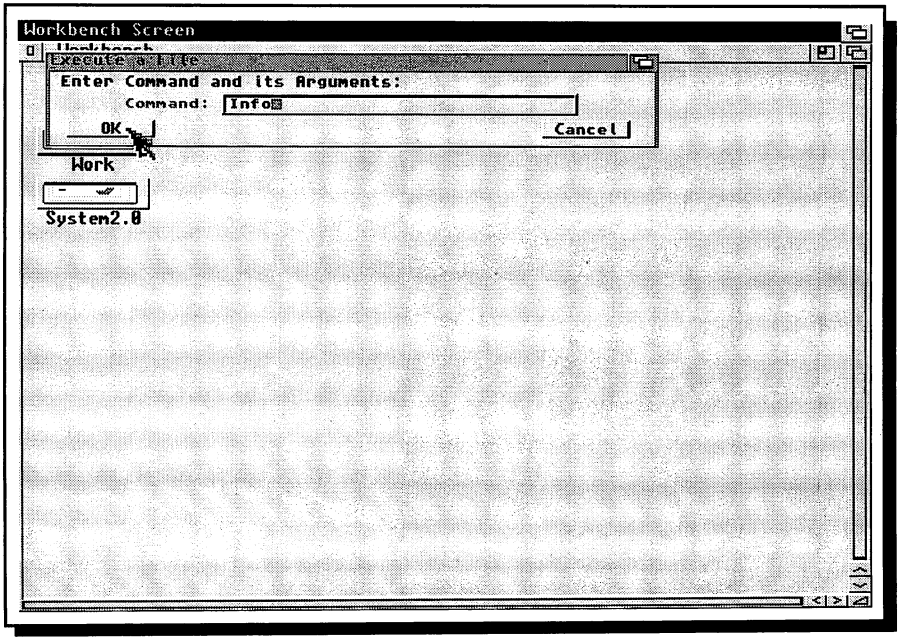


Figure 3-2 Execute Command

The Execute Command item brings up the Execute a File requester. Here, you enter the name of the AmigaDOS command, then press the Return key or click on the OK button to perform the command.

The commands you execute using this menu item are AmigaDOS commands that you will learn about in later chapters. This item lets you execute a command without having to open the Shell. For now, as a demonstration, choose the Execute Command item and enter

INFO

into the requester and select the OK button. (Note: Pressing the Return key after typing Info has the same effect as selecting the OK button.) If you entered a legal command, Workbench opens the Workbench output window and displays the result of the command. The Info command (see Figure 3-3) displays information about all the physical disk drives mounted on your system, giving their AmigaDOS device names, sizes, usage information, and Workbench volume names. To close the Workbench Output Window, either select its close gadget or type CTRL-\ when the window is selected. Obviously, you will be able to make better use of Execute Command once you are familiar with the AmigaDOS commands.

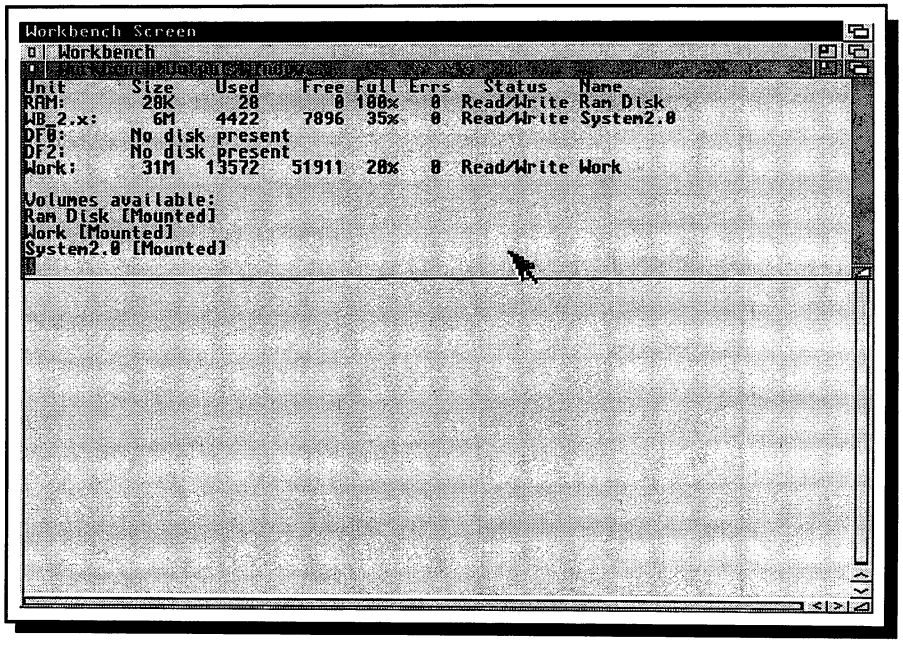


Figure 3-3 Workbench Output Window

When you use Execute Command, Workbench creates a temporary window to hold the command's output. Here, the window holds the output of the AmigaDOS Info command, which returns information about the disk drives and other devices attached to the system. You can have multiple output windows open at once.

Redraw All: Called simply Redraw on prior versions of Workbench, Redraw All forces Workbench to redraw all the open windows and rectifies problems with the refresh mechanism.

Refresh is a procedure by which the operating system or an individual program redraws its output display after a window has been uncovered. Dozens of windows can be open on Workbench, yet only those on top are visible on your display. When you close one of these top windows and uncover another, Workbench must refresh the display — update it — to show the contents of the newly revealed window.

Occasionally, the behavior of one or more programs with windows open on the Workbench screen disrupts the normal Workbench refresh process. One type of window is refreshed automatically by the Amiga whenever it is uncovered. Another type holds the program that created it responsible for refresh. Based upon speed and efficiency considerations, programmers choose the type of window that best fits their needs. Occasionally, a program goes awry while refreshing its window. Commodore supplies Redraw All so you can manually force Workbench to rethink and redraw the display. How often will you use Redraw All? In five years, I've used the old Redraw item no more than half a dozen times, and I've yet to need Redraw All when running under version 2.0. Your experience may differ.

Update All: The cousin to Redraw All, this item is useful if you use both the Workbench and the Shell interfaces. Update All tells Workbench to redraw any currently opened windows by rereading the information about the window from disk. If, for example, both the System and Utilities drawers of your Workbench 2.0 disk are open, choosing Update All forces Workbench to reread the icons on your disk so that the display matches the current state of the files on the disk.

Why wouldn't the display match the files on disk? After all, if you create a drawer using the New Drawer item of the Windows menu, the icon for the drawer pops up immediately in the window. Conversely the system removes a deleted file's icon from the window immediately. If Workbench updates its windows with file changes automatically, why bother with an Update All item?

Workbench, however, is not the only means of creating, moving, and deleting files on an Amiga. You can do that and more with the command-line Shell interface. The problem is that the Shell and Workbench do not communicate directly about the changes each interface makes to the contents of a disk. Consequently, if in the Shell you create or delete files on a disk or a drawer whose window is concurrently open, the changes will not be reflected on the Work-

bench display until you close and reopen the window, or until you choose Update All. While Redraw All tries to recreate the display to reflect the current state of the Workbench windows as they exist in memory, Update All recreates the display to reflect the current state of the disks and drawers on disk.

To demonstrate Update All, open both your Workbench2.0 disk and the Ram Disk. Move the windows so that they don't overlap. Now, choose the Execute Command item and, in the string requester, enter:

```
COPY SYS:Shell.info TO RAM:
```

Now, press Return or select OK.

A Workbench output window pops up, informing you that:

```
Shell.info..copied
```

Close the Workbench output window and check the Ram Disk window. Although you just copied the Shell icon file to the Ram Disk, the icon does not appear in that window. To see the icon, choose Update All.

Note that the icon did not appear until you forced Workbench to reexamine the contents of the Ram Disk. Even though Execute Command is a function available through Workbench, it actually operates by opening a limited version of the Shell. Thus, Workbench is as much in the dark about what Execute Command is doing to disk files as it is about what the Shell is doing.

(By the way, I'll explain the meaning of those cryptic items such as Sys: and RAM: in the chapters on the Shell.)

Last Error: This being an imperfect world, the Workbench program sometimes cannot perform a function you request. When this happens, AmigaDOS generates an error code and sends it back to Workbench. The Workbench then reports the error condition as a requester or as an error line on the menu bar. For example, when you try to format a disk that is write-protected, Workbench puts up a requester telling you that the disk is write-protected. You can then cancel the formatting operation or change the position of the write-protect tab and continue. At other times, or even in conjunction with a requester, Workbench writes an error message to the menu bar. This message remains visible until you press the select button. To see the message again after it has disappeared, you use the Last Error item.

To see an example of the latter case, open your system disk and try to drag the Trashcan icon out of the window. (Your system disk is the disk you used to boot your Amiga; normally either Workbench2.0 or System2.0. From now on, I'll use the term system disk to refer to either of these volumes.) You'll get an

error message in the menu bar saying that it can't be moved out of its window (see Figure 3-4). If you now click the selection button, you can still see the message by accessing the Last Error item.

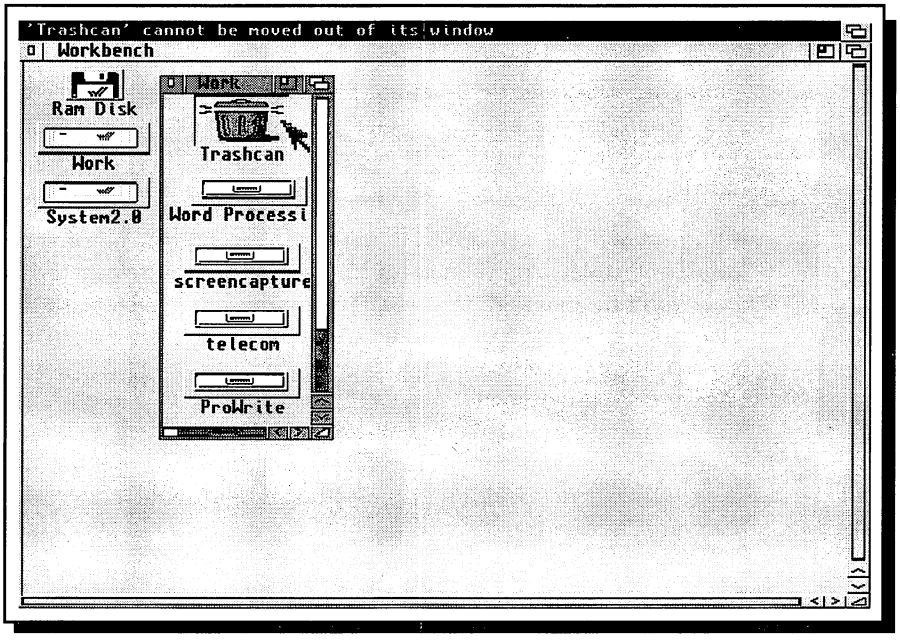


Figure 3-4 Last Error Message

Trying to move a trashcan out of its window results in the above error message. After you've erased the message by hitting the selection button, you can always call it back with the Last Error item.

Last Error isn't for error conditions only. It sometimes simply displays an informational item. For example, whenever you load a program using Workbench and then check the Last Error item, the menu bar will display a message that Workbench is attempting to load the program.

Version: Here is a menu item that does just what you expect. When you access the Version item, Workbench flashes its screen and prints the version of Workbench and Kickstart currently active in your Amiga. You may have to check this item if you buy a program that requires a particular version of Workbench or Kickstart. Usually, software will state on the package which version of AmigaDOS or the Amiga OS (1.2, 1.2 or later, 1.3, 2.0, and so on) the software requires.

Quit: Choosing Quit lets you shut down the Workbench interface. Before you can quit Workbench, however, you must close all programs that you launched from Workbench. This includes both programs that run on the Workbench screen and those that open their own custom screens. If you access the Quit item before you've shut down all Workbench-launched programs, Workbench displays a message in the menu bar telling you how many Workbench programs are still active. If all Workbench-launched programs have been closed, the system puts up a requester that gives you one more chance to cancel the Quit command. If you select OK from this requester, Workbench will shut down.

Quitting Workbench is like sawing the steering wheel off of your car; you can do it, but, unless you've made arrangements for another method of control, you won't be able to do much. Once you're comfortable with the Shell interface, you can safely quit Workbench as long as you have a Shell open on the screen. Note that any Shell you launch from Workbench is considered a Workbench program and must be closed before you can quit. Only Shells opened using Execute Command or from another Shell will remain open after you access the Quit item in the Workbench menu. If you mistakenly shut down Workbench without leaving an open Shell, you'll have to reboot your computer before you can use it again.

The Window Menu

Unlike those in the Workbench menu, the nine items in the Window menu work only with the currently active window. This includes the Workbench window itself, although, because of its special nature, not all Window menu items work on the Workbench window.

New Drawer: This first item creates a new drawer in the currently selected window. Let's take a look: Open the Utilities drawer on your Workbench2.0 or System2.0 disk and activate the New Drawer item. Immediately, Workbench creates a drawer in the Utilities window with the name Unnamed1. Workbench also provides a requester (see Figure 3-5) that prompts you to enter a new name for the drawer. If you simply click on the OK button, the drawer will keep the name Unnamed1. Because this isn't very descriptive, you should use the text gadget in the requester to give the drawer a name — such as Test — before you click OK. That's all it takes to create a drawer under Workbench. If you double-click on the new icon, Workbench opens a standard window for the drawer. The window contains all the usual Workbench gadgets. Because it is a standard Workbench window, you can even create new drawers in this new drawer.

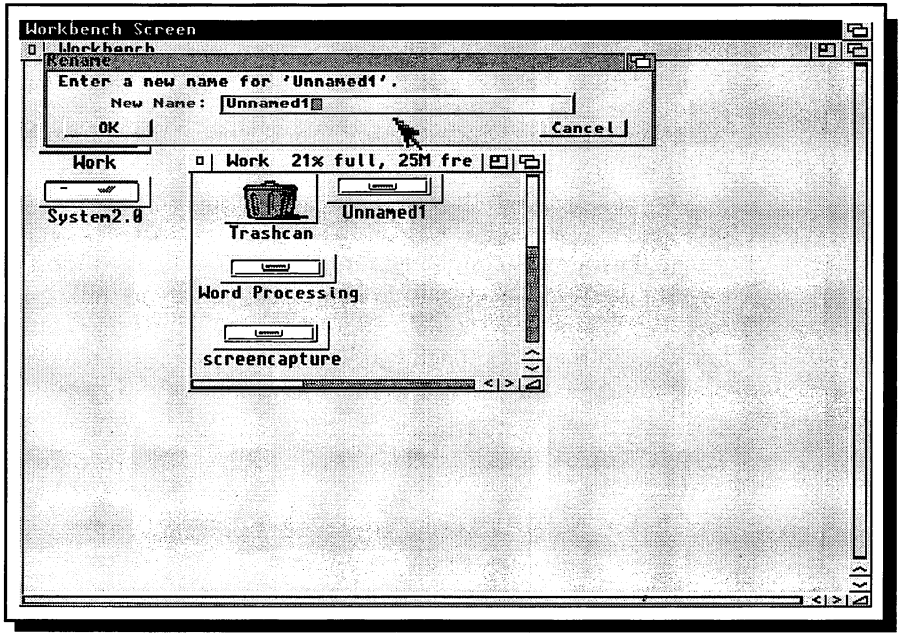


Figure 3-5 Naming a New Drawer

The New Drawer item not only creates a new drawer in the active window, it also calls up the Rename requester with the default name of new drawers — Unnamed1 — in the string gadget. If you choose to keep the default name, subsequent drawers will be called Unnamed2, Unnamed3, and so on.

The only window you can't create drawers in is the main Workbench window. Thus, when the main Workbench window is selected — even if it is a backdrop window — the New Drawer item is ghosted. Because drawers represent a storage area on a disk, they must always be subordinate to a particular volume. Therefore, you can't create a drawer on the Workbench; a new drawer can only exist within an already extant drawer or volume.

New Drawer is a very important item. By letting you create new drawers, Workbench gives you the means to organize your disks to suit your needs. Let's say, for example, that you use your Amiga for three primary applications — word processing, graphics, and telecommunications. To keep your programs and data files organized, you should create one drawer for each of your applications. You can create drawers within these main drawers to hold different kinds of files.

Open Parent: Many times, when you have a dozen or more windows open at once on Workbench, you may have a hard time finding a particular window. Open Parent makes this easier.

Except for the main window, every Workbench window has a parent. The parent window contains the icon that you used to open the current window. For example, the Prefs drawer icon that corresponds to the Prefs window is found in your Workbench2.0 (or System2.0) disk. Consequently, the Workbench2.0 window is considered the parent of the Prefs window. Thus, windows on the Workbench exist in a hierarchy of parents and children. At the top is the Workbench window. Next are the windows that correspond to the disk icons in the Workbench window. Finally, come the windows that correspond to the various drawers in the disk windows (see Figure 3-6).

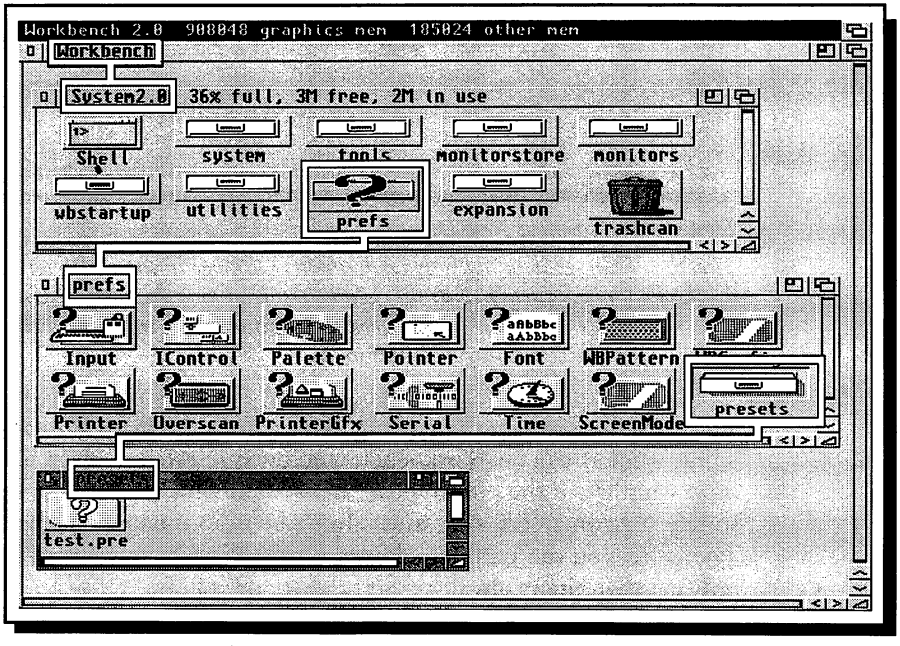


Figure 3-6 A Hierarchy of Windows

Maneuvering around Workbench windows is like climbing an upside-down tree. At the top is the trunk — the Workbench window. You then follow a path of the various branches — disk and drawer windows — down until you reach the tool or project you want. Files in different windows can have the same name as long as they have unique pathnames.

When you open a window, you are actually moving down this hierarchical structure — moving from parent to child. The Open Parent item lets you move in the opposite direction, from the child to the parent. It doesn't close the child window, but it does open the parent window, if necessary, and moves it to the front and makes it active. To see how this item works, open the Prefs icon in your Workbench2.0 (or System2.0) disk. Now, close the disk window and select the Prefs window. Next, activate Open Parent. The system opens the disk window — the parent to Prefs — automatically and moves it to the front of the display. If you access Open Parent again, the main Workbench window — the parent to the disk window — pops to the front and becomes active. Because this window normally covers the entire screen, bringing it to the front will cover all the other open windows on the display. (Note that Open Parent will not bring the main Workbench window to the front if you have chosen the Backdrop option.)

Close: This item is simple: It closes the currently selected window. It has the same effect as clicking on the close gadget in the upper-left corner of the window. The only exception is for the main Workbench window. In this case, Close works the same as the Quit item in the Workbench menu, closing the Workbench if all Workbench-launched programs are shut down.

Update: This item works identically to the Update All item in the Workbench menu, with the exception that it works in the currently selected window only. It rereads the disk area that corresponds to the current window and updates the icons according to the current information on disk, taking into account any changes made in the Shell of which Workbench was unaware. Update works only on the current window. It has no affect on either the parent or children of the current window.

Select Contents: Yet another way to select multiple icons on the Amiga, Select Contents selects every icon in the active window. It does not affect the parent or children of the window. Like the shift-select and drag-select functions described earlier, it lets you perform the same action — such as delete — on multiple icons. It automatically deactivates any icon you previously selected.

Clean Up: Unlike the workbench in your basement, the Amiga Workbench gives you a quick and easy way to clean your work area. Choosing the Clean Up item invokes a Workbench routine that rearranges the icons in the active window so that they appear in orderly rows and columns (see Figure 3-7). Starting with the upper-left corner of the window, it moves the icons so that they fill the first column, then the second, and so on, until all the icons are rearranged. Clean Up always fills columns before rows. The effects of Clean Up are not permanent. If you close and reopen a window you cleaned up, the icons will appear in their original, messy state.

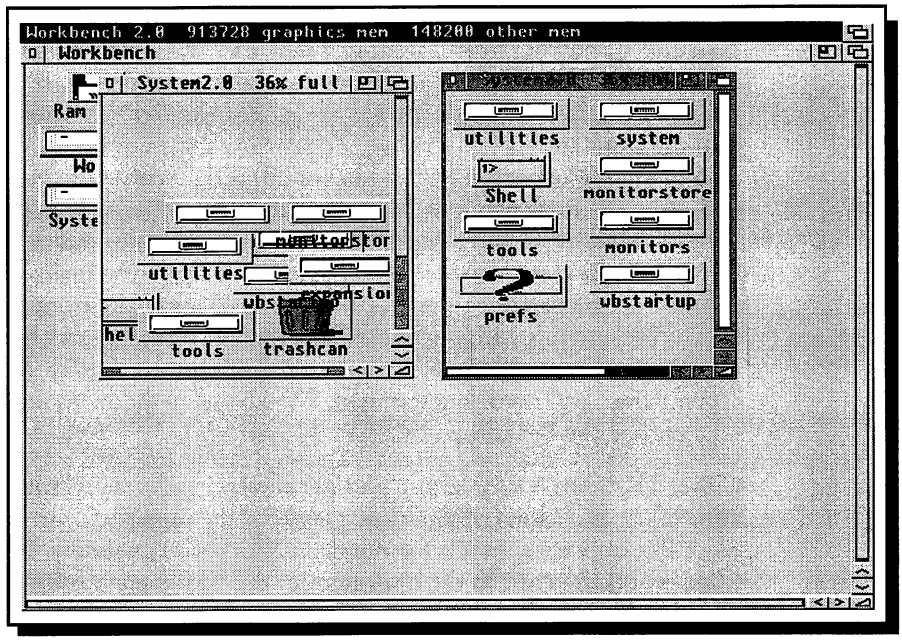


Figure 3-7 Clean Up

The Window on the left needs the help of the Clean Up option. The window on the right shows its results. Don't you wish you could use this option on your closets!

On previous versions of Workbench, Clean Up would work only if it was the first action you took after opening a window. With Workbench 2.0, you can invoke Clean Up anytime you have a window selected.

Snapshot: Up until now, all the menu items you've encountered have been simple menu items; they have performed one function only. Snapshot features a submenu that lets you choose from two options — Window and All. You can distinguish a simple menu item from one with a submenu by the presence of two right-hand angle brackets (>>) to the right of latter's name in the menu. When you move the pointer over an item with a submenu, the submenu appears to the right. To select from the submenu, move the pointer over the function you want and release the button, as for a standard menu (see Figure 3-8).

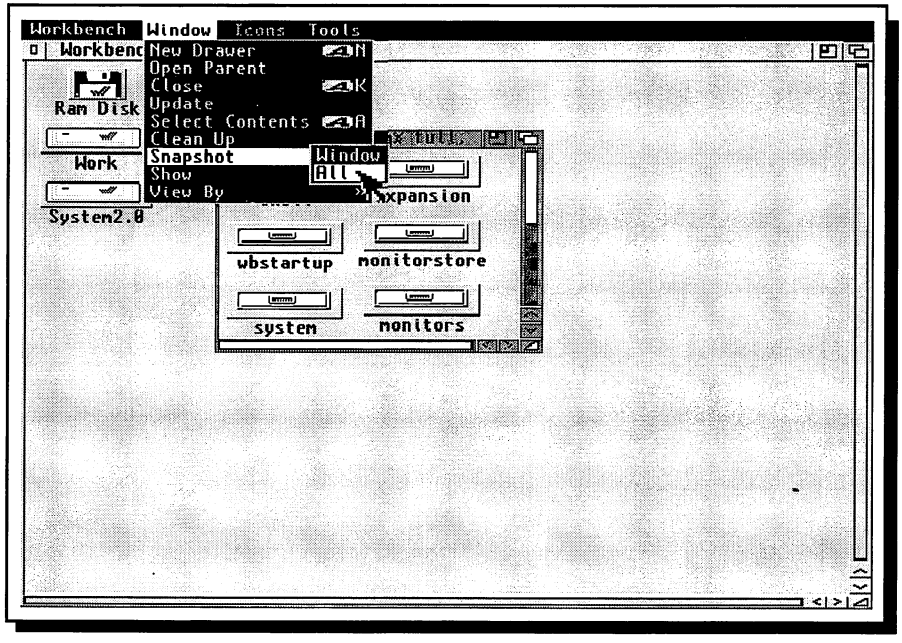


Figure 3-8 Submenus

The Snapshot item is one that uses a submenu. Keeping the menu button down, you can highlight the submenu items as you would regular menu items.

Snapshot Window: Choosing this item saves the current position and dimensions of the active window. When you next open this window, even if you have subsequently moved or resized it, it will appear in the same place and at the same size as it was when you took the snapshot. Snapshot Window works only on the window itself; it doesn't affect the icons within windows.

Snapshot All: This item saves position information on both the active window and its icons. It is the natural adjunct to the Clean Up item, because it lets you make permanent that function's changes. For the most part, whenever you access Clean Up, the next thing you should do is access Snapshot All. Note: Snapshot works on true icon files (.info files) only. It does not work on the pseudo-icons created with the Show All Files item discussed below.

Show: Through version 1.3 of Amiga OS, you could not view files without complementary .info files from Workbench. Because the information Workbench needs to draw icons is contained in .info files, such files never appeared on the Workbench interface. The Show item changes that. It gives you the

option of seeing all the files on your disks with the Workbench interface. The Show submenu has two items: Only Icons and All Files.

Show Only Icons: This is the default setting for Workbench. With this option active only files that have a corresponding .info file will appear in any window you open on Workbench.

Show All Files: Choosing this item causes Workbench to display both files that have a corresponding .info file and files that don't. For files without .info files, Workbench uses default icon information that it stores internally. Figure 3-9 shows these default icons. Note that because the default icons are not .info files, you can't take snapshots of them as you can true icons.

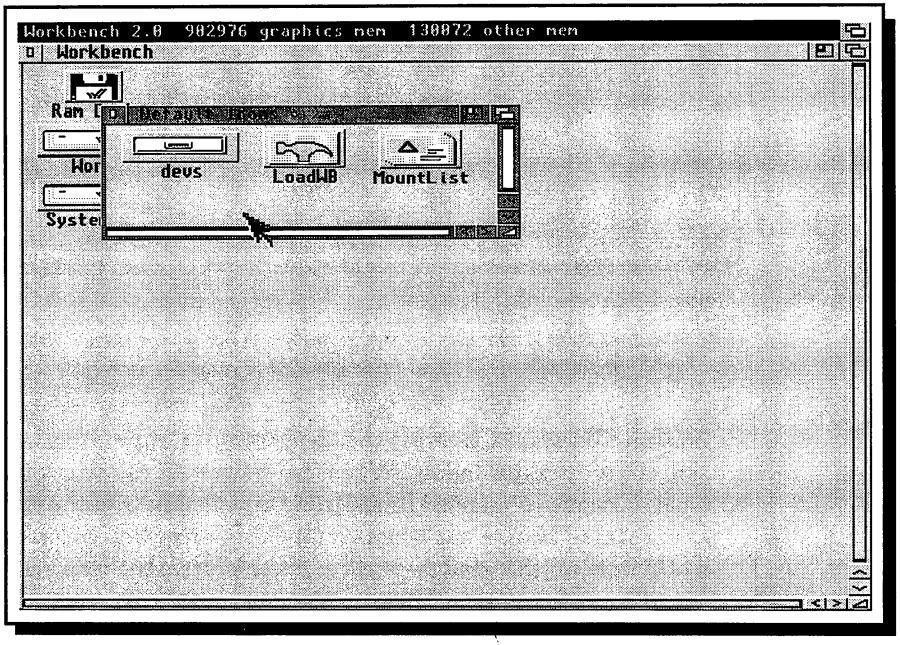


Figure 3-9 Workbench Pseudo-Icons

When you choose Show All, Workbench uses default images for files that don't have .info files. The drawer image is identical to that used by other drawers on the Workbench 2.0 system disk. Also shown are the default tool and project icons.

To see how Show All Files works, open your Workbench2.0 (or System2.0) disk. Now, with the disk window active, choose Show All Files. Immediately, seven new drawers appear: Devs, C, L, S, Rexxc, Fonts, and Libs (see Figure 3-10). These drawers, normally invisible because you work with their contents primarily through the Shell, are now available to Workbench.

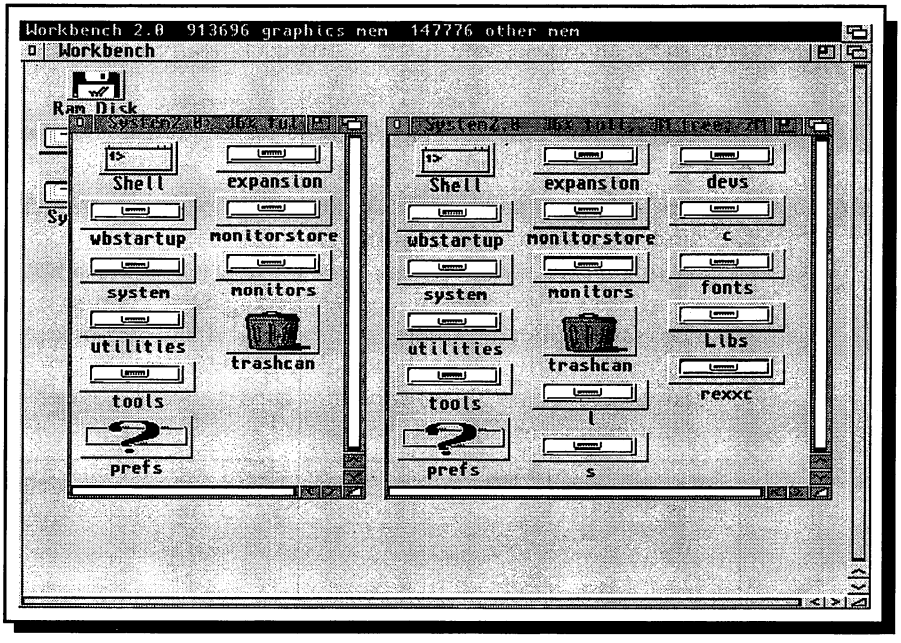


Figure 3-10 System2.0 Window Using Show All

The left image is the System2.0 window as it normally appears. On the right is how it appears after you choose Show All. The drawers that appear are very important for working with the Shell interface.

Unlike many other items, Show affects child windows as well as the active window. From the example above, open the C drawer. None of the items in this window have icons, yet they will all be visible as long as you've selected Show All Files in the parent window. You can override the setting of a parent window by selecting the Show option you want explicitly for the active window.

Workbench 2.0 not only lets you see files without .info files, it also lets you perform many functions with them that you can with standard Workbench files. To demonstrate, move the window of the C drawer to the upper-left corner of the Workbench screen and resize it to fill the Workbench screen. Now, double click on the icon called Dir. Workbench will display a requester identical to that used by the Execute Command item, except that the command Dir is already in the text gadget. When you select the OK button, the system puts up a Workbench output window that displays the results of the command. In this case, it lists the contents of the C drawer.

View By: The last item, the Window menu, also uses a submenu. View By lets you choose how you want to see Workbench icons in a window. The options are Icon, Name, Date, and Size.

View By Icon: This is the Workbench default. When you open a Workbench window, the system displays the icons of the files in the window.

View By Name: Selecting this item displays an alphabetical list of the current window's contents. You can still manipulate the files with the mouse by clicking on the name, but the icons are invisible. For example, you can open the window of a drawer by double-clicking on the drawer name in the list. The Name option also displays the sizes of the files in bytes, their protection bits, and the date and time each file was created (see Figure 3-11). Note that drawers don't have a size listed; they have the word "drawer" in the size field to distinguish them from other files.

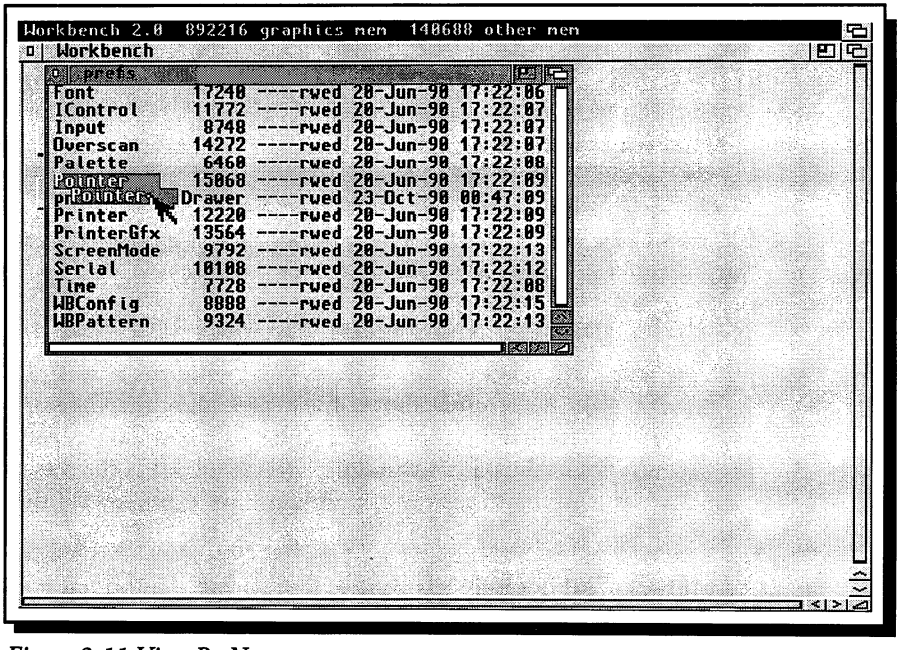


Figure 3-11 View By Name

This item gives more information about Workbench files than do conventional icons. From left, the fields are filename, size ("drawer" if the file is a drawer), protection bits, and date and time of creation. Note that you can select and manipulate a filename just as you can an icon.

View By Date: This item also lists the file in the window by name, but it sorts them in chronological order instead of alphabetical order. It lists older files first and newer ones last.

View By Size: This option lists the files in a Window by name, as sorted by their size in bytes. Drawers are considered to be zero bytes in length. Files of equal length are also sorted alphabetically.

The Icons Menu

The third Workbench menu — Icons — provides facilities that let you manipulate individual files and groups of files. You indicate the files you want to work with by selecting the appropriate icons. As you saw earlier, you can select icons in numerous ways: clicking with the select button, drag selecting, shift selecting, and using the Select Contents item from the Window menu. Until you've selected one or more icons, the Icons menu remains ghosted and inaccessible.

With one or more icons selected, the Icons menu makes 11 items available to you — Open, Copy, Rename, Information, Snapshot, Unsnapshot, Leave Out, Put Away, Delete, Format Disk, and Empty Trash.

Open: As you might suspect, choosing the Open item opens all selected icons. The confusion with this item comes from the fact that the Amiga supports a half dozen different types of icons, and the word "open" means something different to each icon type. In all cases, however, Open performs the same function as double-clicking on the icon with the selection button.

Tool: Because a tool icon normally represents an executable program, opening one will execute the corresponding program. The simplest example of this is the Clock icon in your boot disk's Utilities drawer. If you select the Clock icon and choose Open from Icons, you tell Workbench to load and run the Clock program.

One anomaly of the Open command is evident when you use the Show All option from the Windows menu. Workbench uses its default tool icon for any iconless file that has its Executable or Script protection bit set. (Workbench uses protection bits to describe files and indicate the operations you can and cannot perform on them. Protection bits are discussed in the Information item section below.) Thus, accessing the Open item lets you execute scripts as well as programs.

(A note of caution: Be careful not to open the icon of the Startup-sequence file in the S directory. This script should only be executed when you boot

your system. Executing it at any other time is a sure-fire way to crash your machine.)

Drawer: With a drawer icon selected, Open tells Workbench to open the drawer's window.

Disk: As with drawer icons, the Open item opens the window of a selected disk icon.

Project: Because projects are created by tools, choosing Open when a project is selected first opens the tool that created the project and then loads the project into the tool. For example, when you open a picture file (the project) that was created with DeluxePaint (the tool), Workbench finds and loads the DeluxePaint program, then loads the picture file into it. Note that the project icon must pass information about the appropriate tool's location to Workbench. If Workbench can't find the tool, the system displays a requester saying that Workbench is unable to open the tool.

When a tool creates a project, it invariably inscribes its location into the project's icon. If you move the tool, however, the location in the project icon is no longer valid. The Information item lets you provide a project with the tool's new location.

Projects created with tools like DeluxePaint and ProWrite and other commercial programs have their Executable protection bits set. Even though these projects are not programs, opening them results in the execution of a program. When you use the Show All option, Workbench uses its default project icon for any file that doesn't have its Executable bit set. Thus, after using Show All, you will run into projects that do not cause Workbench to execute a tool when you open them. Check out the projects in the Devs drawer of your system disk for examples. Opening the Mountlist project, for example, brings up the Execute Command requester. Clicking the OK button merely brings up the Workbench output window with the message that Mountlist is not an executable file.

Trashcan: The final icon type used by Workbench is the trashcan type. (While trashcan icons are normally named Trashcan, you can rename them.) A trashcan icon is a special type of drawer. Its capabilities and uses are discussed below in the Empty Trash item section.

Why did Commodore provide the Open option for icons when double-clicking them is much easier? Open can work on multiple icons at once. If you select four or five drawer icons and access Open, Workbench will open the windows of each drawer. The same does not, however, hold true for executable files. Accessing the Open option with multiple executable icons selected only runs the file of the first icon selected.

Copy: In the last chapter, you learned that dragging an icon between different volumes — or the subordinate windows of different volumes — made a copy of the file on the destination disk. The Copy menu item lets you make duplicates of a file in the same window as the original. It will copy as many items as you've selected.

Copy works not only on programs and projects, but also on drawers, trashcans, and disks. In all cases, however, the mechanism is the same. To duplicate an object, you select it and then choose Copy from the menu. Workbench will make a copy of the icon (and its associated file, of course). The new file will have the name Copy_of_xxx, where xxx is the name of the original file. If you subsequently copy either the original or the duplicate, the new file will have the name Copy_2_of_xxx. Subsequent copies will be called Copy_3_of_xxx and Copy_4_of_xxx, and so on. (It's a good idea to use the Rename item after you've made a copy of something.) Copying a trashcan, however, is a waste of time. Because all trashcans perform the same function, and because they can't be moved out of a window, it doesn't make much sense to copy one and wind up with two in a single window.

Copying drawers can be a tricky proposition. The Copy item not only copies the drawer but also the contents of the drawer. If you copy a drawer that has five levels of drawers containing 300 projects and tools, the Copy item will duplicate them all. Be careful copying drawers, lest you get more than you bargained for.

Copying disks is a special case. To copy a disk, Workbench calls upon the DiskCopy tool found in the System drawer of your system disk. (Workbench also uses DiskCopy if, in a two-drive system, you drag the icon for one disk on top of another.) If Workbench can't find the DiskCopy tool, it will abort the operation.

Once you've selected a disk icon and chosen the Copy item, DiskCopy puts a window and a requester on the screen. The requester tells you to put the source disk in DFx:, where x is a number from 0 to 3. "Source disk" is what Workbench calls the disk you want to copy, and DFx: is the AmigaDOS name of the disk drive the source disk is in. If you have one disk drive, it is called DF0:. (Depending upon the model of your Amiga, your second drive may be DF1: or DF2:. Consult your manual or use the Info command with the Execute Command menu item, as I demonstrated earlier.)

Write protecting the source disk is a good idea at this point. You write protect an Amiga floppy disk by moving the write-protect tab, located at the upper-left of the back of the disk, to the inhibit position. A disk is write-protected if you

can see through the hole below the tab. A disk is write-enabled if the tab covers the hole. With the disk write-protected, you're ready to proceed.

If the source disk isn't in the drive, put it there before you click the Continue button on the requester. (You can click Cancel if you want to abort the process.) Next, DiskCopy begins reading the source disk. On my Amiga 3000, it reads half the disk before stopping. Once it stops reading the disk, the program prompts you to put the destination disk into the drive. Before ejecting the source disk, be certain that the disk-activity light is out. Pushing the eject button while the disk is still spinning will corrupt the disk and perhaps harm the drive.

Now, remove the source disk and pop in the destination disk — the disk to which you are copying. This can be a blank, unformatted disk or a previously used volume. Note that the disk-copying procedure destroys all information on the destination disk and replaces it with a copy of that on the source disk. When you click the Continue button, DiskCopy writes what it previously read to the destination disk. When it is finished writing, it again prompts you to insert the source disk. It continues to prompt you to swap disks until the copy is complete. As with copying other objects, the new disk is called Copy_of_xxx, where xxx is the name of the source disk. If memory is limited on your machine, copying a disk may take more than two swaps. Shutting down unnecessary programs and windows will free up memory.

Last thoughts on copying disks: The Copy menu item uses only one disk drive, even if you have two or more on your system. With two or more drives, you can save time by putting the source and destination disks in different drives and dragging the icon of the source over the icon of the destination.

Rename: The Rename function lets you change the name of an icon, thereby changing the name of a file. When you use Rename, Workbench pops up a requester displaying the current name of the icon in a text gadget. By changing the name in the gadget, you change the name of the icon. After you type the new name, you select the OK button to make the change or the Cancel button to retain the old name. Rename works on both the .info file and the file it represents. For example, if you rename the Clock icon to Timepiece, its .info file gets renamed to Timepiece.info.

While Rename is very helpful for organizing your files and drawers, you should not use it to rename tools or important system files. If you rename tools, Workbench won't be able to find them when you open a project. The same goes for system files. For example, AmigaDOS expects your startup script to be called Startup-sequence. If you rename this file, AmigaDOS won't be able to execute it.

AmigaDOS has few restrictions on what you can name a file. Filenames can be up to 31 characters long and contain numerals, letters, and special characters such as "%" and "*". If you plan to use the Shell to augment or replace the Workbench interface, however, you should be careful to avoid filenames that contain spaces. Although such names are permitted under AmigaDOS and the Shell, they are harder to work with than names without spaces.

You don't have to worry about giving two files the same name. As long as the files are not in the same window, they can have the same name. Workbench can keep them apart because it uses a file's complete pathname, not just its filename, to identify a file. The pathname consists of the filename and all the names of the drawers and disks above it in the hierarchy. Thus, files with the same name in different drawers will have different pathnames.

Information: One of the more complicated items on the Workbench, Information is also one of the most important. The name is self-explanatory: Information gives you information about icons. The type of information depends upon the type of icon you are accessing.

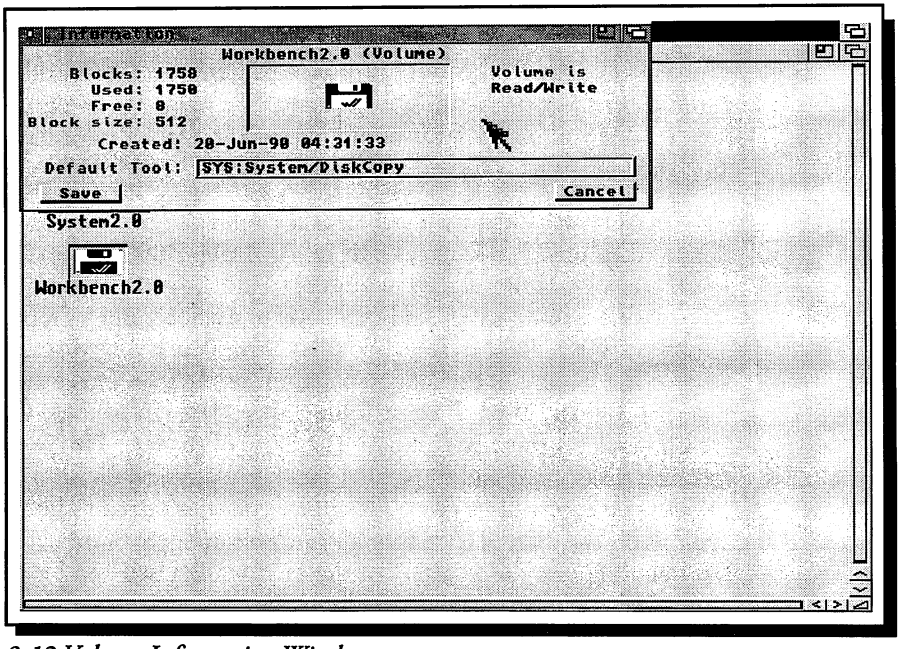
Disk: Consider Figure 3-12. This is the information window for the System2.0 volume that came as part of my Amiga 3000 system. The window displays a lot of useful data about the volume. At the top-center of the Information window, you find the name of the icon and its type; in this case, System2.0 is a volume, which is what Workbench calls disks. Below the name is a picture of the icon. To the left of the icon image are four numbers labeled Blocks, Used, Free, and Block size. The last of these tells you that each block on the disk contains 512 bytes of storage. This is standard for all Amiga disks, hard and floppy. (A block is a physical division on a disk. Because you access disks using filenames, you don't need to know much more about blocks than that each holds one half kilobyte of data.) The Block number reports the total size of the disk, while Used and Free state the number of blocks currently used to store files and the number available to store files, respectively. Used and Free always add up to the number in Block.

To the right of the icon image is the read-write status of the volume. A volume can be either read-write, allowing you to load files from and save files to the disk, or read-only, letting you only load from the disk. You can change the read-write status of a floppy disk by moving the read-write (write-protect) tab on the disk.

Below the image is the date and time the volume was formatted. This information is only as accurate as your Amiga's internal clock was when you created the volume. Below this is a text gadget that lists the icon's default tool. In the case of a volume icon, this is the DiskCopy tool in the System

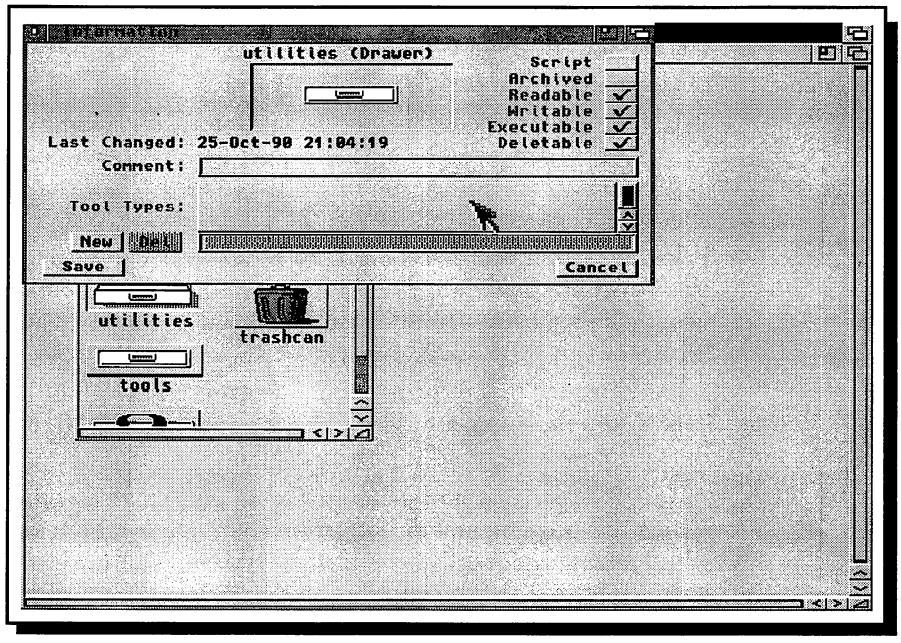
drawer. Volumes are not executable files, so the default tool has little meaning. Because the default tool is listed in a text gadget, you can change it if you like. Unlike project icons, however, you'll probably never have reason to change the default tool of a volume icon. The Save and Cancel buttons at the bottom of the Information window let you keep or discard changes you've made to the icon's default tool.

Drawer: The Information window for a drawer icon differs significantly from that of a volume icon. The display for the Utilities drawer of you system disk is shown in Figure 3-13. Unlike disks, drawers do not have a fixed size, so the information display does not show information about the size of the drawer. The second difference is the presence of protection bits.



3-12 Volume Information Window

Perhaps the most important information you need about a disk is the amount of storage space left. The Information item lists this data in blocks, which are equal to 512 bytes. Unlike other types of icons, volumes don't have protection bits.



3-13 Information on the Utilities Drawer

The protection bits let you decide which actions can be taken with a drawer or its contents. The Deletable and Writable bits are especially helpful for protecting valuable files from accidental overwriting and erasure.

Protection bits provide Workbench, programs, and you with information about the status and capabilities of an icon. Workbench currently recognizes six protection bits: Script, Archived, Readable, Writable, Executable, Deletable. (AmigaDOS recognizes a seventh bit, the pure bit, which Workbench doesn't use.) Protection bits are actually two-state indicators or flags: They tell you whether or not a file currently has a particular status or attribute. In the Information window, Workbench puts a check next to the attributes that are set for the current file. If the Script bit is set, the file contains an AmigaDOS command script — a sequence of AmigaDOS commands. If you enter the file name on the Shell command line or in the Execute Command window, AmigaDOS recognizes it as a script. The Archived bit is used by some backup programs to indicate that a file has been backed up. The Readable bit indicates whether a file can be read from disk, while the Writable bit indicates that you can overwrite the file with a newer version. The Executable bit indicates whether a file is executable, while the Deletable bit lets you protect a file from deletion, if you wish. You

can change the status of the protection bits by clicking on the button next to each one. To preserve your changes, you must click the Save button in the lower-left of the window.

Below the icon image is the Last Changed line, which indicates when the drawer was created, and below that comes the Comment line — a string gadget you can use to store up to 79 characters that describe the contents and purpose of the drawer. Use of the Comment line is optional.

Below the Comment line is the Tool Types requester, which I will discuss below with the Tool information display. The Drawer information window's final two gadgets let you save or cancel any changes you made.

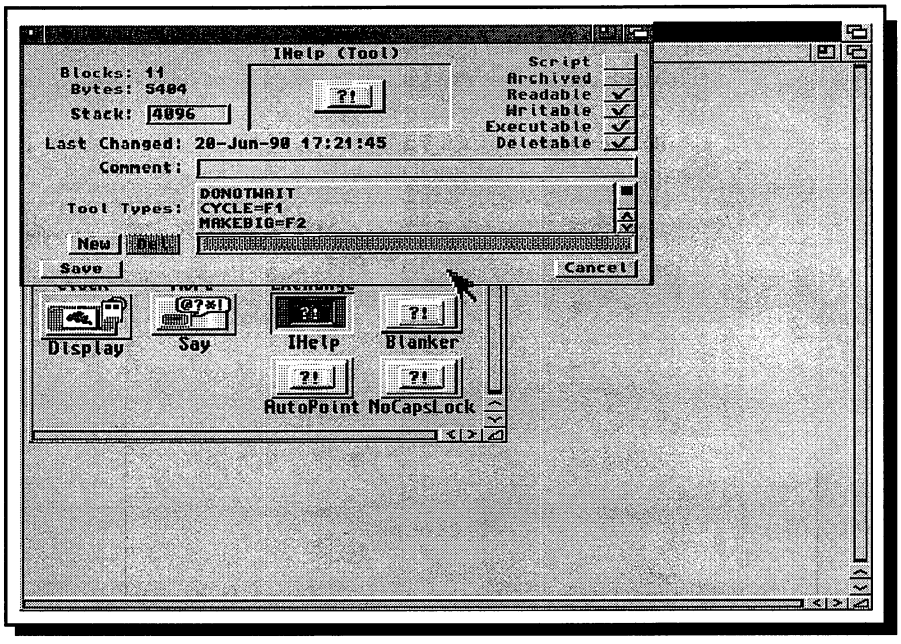
Tool: The Information window for a tool icon adds two elements to that for a drawer icon. The first is the size information to the left of the icon image. This details the number of blocks the file uses on disk and the exact size of the file in bytes. Below this is the second new element, the Stack indicator.

A stack is a special area of memory used by a program and the operating system to store temporary information about the program. For example, when your program calls a routine from the Amiga ROM kernel, it stores parameters needed by the routine on the stack, where in turn the routine looks for them automatically. Each program on the Amiga has its own stack memory, and most Workbench and AmigaDOS programs use a stack of 4,096 bytes. Some, like the AmigaDOS Sort command, require larger stacks. Unless instructed by the a program's documentation, you should not change the value in the Stack gadget.

Perhaps the most important requester in the tool Information window is the Tool Types requester, which consists of a scrollable, three-line display above a one-line string gadget. To the left of the gadget are two buttons, New and Del. Tool Types let you change certain parameters or attributes of a program. Not all programs support them, however, so consult the documentation for each to determine how Tool Types are used. Most of the programs that come with Workbench do not support Tool Types, but those that do give you great flexibility.

To see what I mean, open the Utilities drawer of you system disk and select the IHelp icon. Now, access the Information menu item. You'll see a display similar to that in Figure 3-14. Using the scroll bar or the arrows, you'll discover that the IHelp tool supports six Tool Types. Let's modify one to see how Tool Types work. Scroll to the bottom of the Tool Types display. The line ZIPWINDOW=F5 tells the IHelp tool that its Zipwindow function, which is identical to the function of the zoom gadget, is controlled by the F5 key. Thus, with the IHelp tool active, pressing the F5 key performs the same function on the active window as does clicking on the zoom gad-

get. Now, select the Zipwindow Tool Type: it will appear in the string gadget. Move the cursor to the end of the line and change the F5 to F6. Finally, click on the Save button. When the Information window disappears, double-click on the IHelp icon to activate its functions. Now, press the F6 key. Note that the current window — which should be the Utilities window — reacts as if you clicked the zoom gadget. You've just customized the IHelp tool to use the F6 key instead of F5.



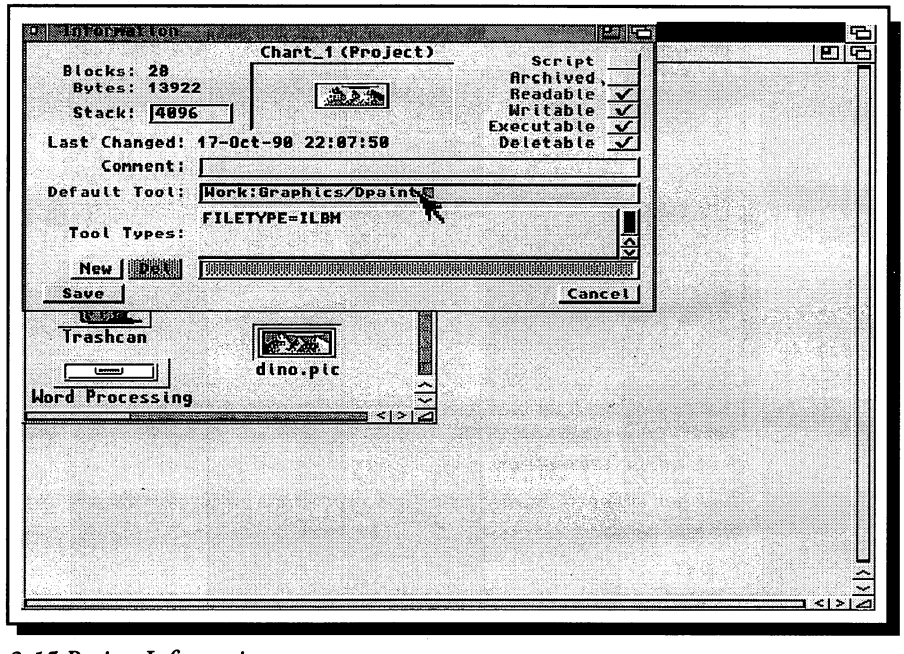
3-14 IHelp Information

This tool supports six different Tool Types, thus giving you the ability to customize how the tool works.

Such a change certainly isn't major, but some of the things you can do with Tool Types are. For example, I'm writing this chapter with the ProWrite word processor, which normally opens a custom, eight-color screen. Eight-color screens take up a lot of memory — about 50 percent more than the default, four-color Workbench screen. Rather than forcing me to close down other programs to make room for it, however, ProWrite provides a Tool Type called WB that lets me modify the program to open on the memory-efficient Workbench screen. Tool Types thus increase the flexibility of the programs that offer them.

Project: Like volume icons, the Information window of a project icon contains a Default Tool string gadget that provides the location of the tool used to create the icon. Thus, when you open the icon, Workbench uses this information to locate and execute the tool.

The location given in the string gadget is in the form of an AmigaDOS path. A path describes a file by pinpointing its location in the hierarchy of disks and drawers on the Workbench. In the example in Figure 3-15, the Default Tool gadget contains the string WORK:GRAPHICS/DPAINT. This means that the file Chart_1 was created by the program DPAINT (DeluxePaint), which resides in the Graphics drawer of the volume named Work. (Note that volume names are always followed by a colon and drawer names are followed by a slash. You'll learn a lot more about paths in the Chapter 7.)



3-15 Project Information

The Information window of this project identifies its file type and the name and location of the tool that created it. If you change the location of the tool, you'll have to change the pathname listed in the Default Tool string gadget if you want to launch the tool by opening the project.

If you move a tool to a different drawer and then open one of its projects, you'll probably get a message from Workbench saying that it's unable to

open your tool. In this case, you'll have to modify the path in the Default Tool gadget to reflect the new location of the tool.

Projects also support Tool Types. Some, like the example in Figure 3-15, use Tool Types to indicate the file type of the project. For example, the file Chart_1 is identified as an ILBM (interleaved bitmap) file, which is the Amiga IFF (interchange file format) standard for picture files. Any tool that supports the ILBM format can load this project file. Other projects support the entire range of Tool Types handled by their parent application. Such projects let you modify a tool for use with a certain project without having to modify the default Tool Types of the tool itself.

Trashcan: The Information window of a Trashcan icon is fairly simple, consisting of its name and type, an image of the icon, its protection bits, a Last Changed line, the Comment gadget, and the Save and Cancel buttons.

As you've seen, the Information item is important not only as a passive information display, but also as a tool to make significant changes in the attributes and functions of many different files. You'll use the Information item often.

Snapshot: Unlike the Snapshot item in the Windows menu, which works on the active window only or on the window and all its contents, the Icons Snapshot works only with selected icons. In this regard, it is similar to the Snapshot option found on older versions of Workbench. Snapshot saves the position in a window of all selected icons. When you next open the window, the icons will appear in the same locations they were in when you accessed Snapshot. Because Clean Up and Snapshot All, in the Windows menu, let you arrange your icons automatically, you may not have much need for this older-style Snapshot. Remember: Both this Snapshot and the one in the Window menu automatically overwrite any previous position information in the .info file.

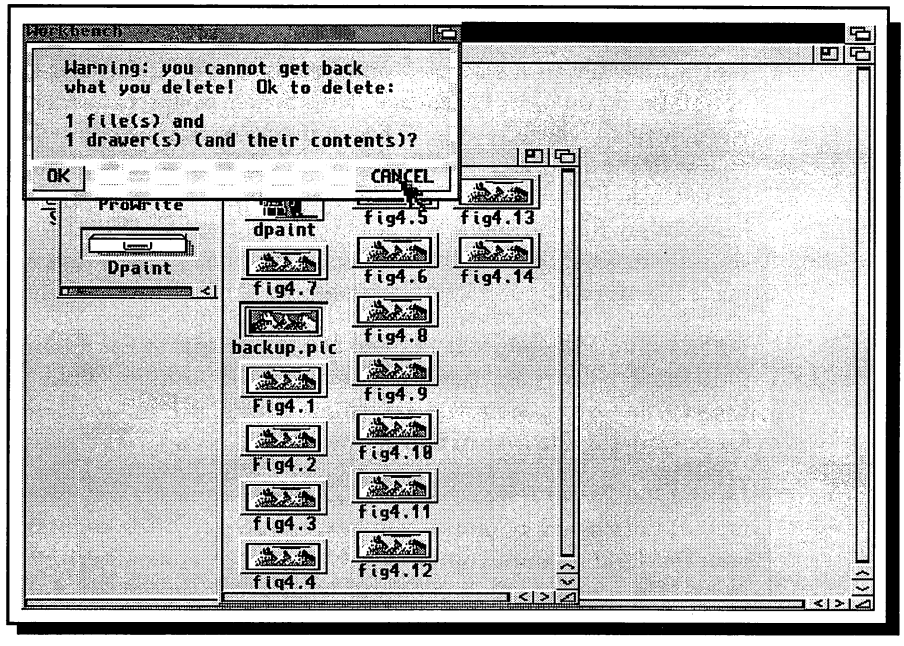
UnSnapshot: The opposite of Snapshot, UnSnapshot sets an indicator in the .info file that clears any previously set snapshot position. When you use UnSnapshot on an icon, you are telling Workbench to use its own algorithms for positioning the icon in its window.

Leave Out: New to Workbench 2.0, Leave Out moves the currently selected icon to the main Workbench window, where it remains — even if you reboot your machine — until you move it back to its original directory with Put Away. The icon functions exactly as before — in fact, it still thinks it is in its original directory. The Leave Out function simply makes it easy for you to access frequently used projects and tools.

Put Away: This item removes an icon from the main Workbench window and returns it to its original directory. It is the complement to the Leave Out item.

Delete: This aptly-named item removes selected icons from their windows and from their disk. Delete works on tools, projects, and drawers but not on disks or trashcans. The fastest way to get rid of unwanted files on your system, Delete eliminates both .info files and their associated files. Be careful: Delete is irrevocable. Workbench does not provide a means to recover what you delete.

When you select an icon and access the Delete item, Workbench puts up a requester listing the number of files and drawers you've selected for deletion and asking if you want to proceed (see Figure 3-16). Always examine the requester thoroughly before you continue, in case it contains surprises. For example, deleting a drawer not only erases the drawer but also all the drawers and files it contains. The requester can alert you to inadvertent selections caused by the extended-selection capability of the shift keys, as well. I discovered this the hard way. To free up space on my disk, I double-clicked on my picture files drawer and, using the shift-select procedure, selected three unwanted files for deletion. I then accessed the Delete item and, without reading the requester, hit the Continue button. You can imaging my horror when all 19 pictures in the drawer, and the drawer itself, vanished from my Workbench and my disk.



3-16 Delete Requester

Before clicking the Continue button, you should check that the number of files and drawers indicated for deletion is reasonable and correct. You can't get back what you delete.

My problem came about because double-clicking on the drawer icon not only opened it but, in the process, also selected it. The drawer icon was never deselected because I used shift-select to select the three files I wanted to delete. (I should have used an unshifted-select for the first file.) The Delete requester did inform me I was about to erase 3 files *and* 1 drawer with all its contents, but I didn't pay any attention to it. Always read requesters!

Format Disk: The Amiga uses standard 3-1/2-inch double-sided, double-density floppy disks. Before you can use a new disk, however, you must format it. Formatting is the process by which AmigaDOS establishes the magnetic signposts that it uses to store and retrieve files on a disk.

When you put an unformatted disk — or a disk formatted under another operating system — into an Amiga disk drive, Workbench displays an icon named DF0:???? or the like (the name depends upon the drive the disk is in). To make the disk usable by the Amiga, select it and choose the Format Disk item. You then get a requester asking you to insert the disk into the drive. If the disk is write-protected, you'll have to write-enable it before continuing. Workbench then informs you that any data on the disk will be erased by the formatting procedure and asks you if you want to continue. Selecting the Continue button starts the formatting procedure, which results in a formatted, ready-to-use disk named Empty.

In addition to formatting blank or alien disks, you can also use Format Disk to erase Amiga disks. Rather than delete everything from a disk you want to reuse, simply reformat it. When you select an Amiga disk for formatting, Workbench puts up a requester with an OK-Quick button. Hitting this button only rewrites the first two blocks of the disk; this has the effect of erasing the disk, because the first two blocks contain the pointers to all the other files on the disk. The quick option is much faster than reformatting a disk from scratch.

Empty Trash: The last item in the Icons menu offers an alternative to Delete. A trashcan is a specialized type of drawer. You can move other drawers and files into it, and access these objects normally. The difference between a standard drawer and a trashcan is the Empty Trash item. When you select a trashcan and access the Empty Trash item, the system immediately deletes all items in the trashcan. Trashcans are a temporary repository for icons before you delete them.

Unlike other drawers, you can never move a trashcan out of a window or delete a trashcan. According to Commodore's documentation, you shouldn't be able to copy a trashcan, but I've had no trouble doing so with my version of Workbench 2.0. Perhaps this will be fixed in a later version.

How useful is the trashcan? I never use it, preferring the quickness and finality of the Delete item. If you're uncertain whether you should delete a particular icon, put it in the trash until you make up your mind.

Tools Menu

Tools, the last Workbench menu, is also the smallest; it has only one item — ResetWB. It has the potential, however, to be the largest, because it is the only menu to which you can *add* items.

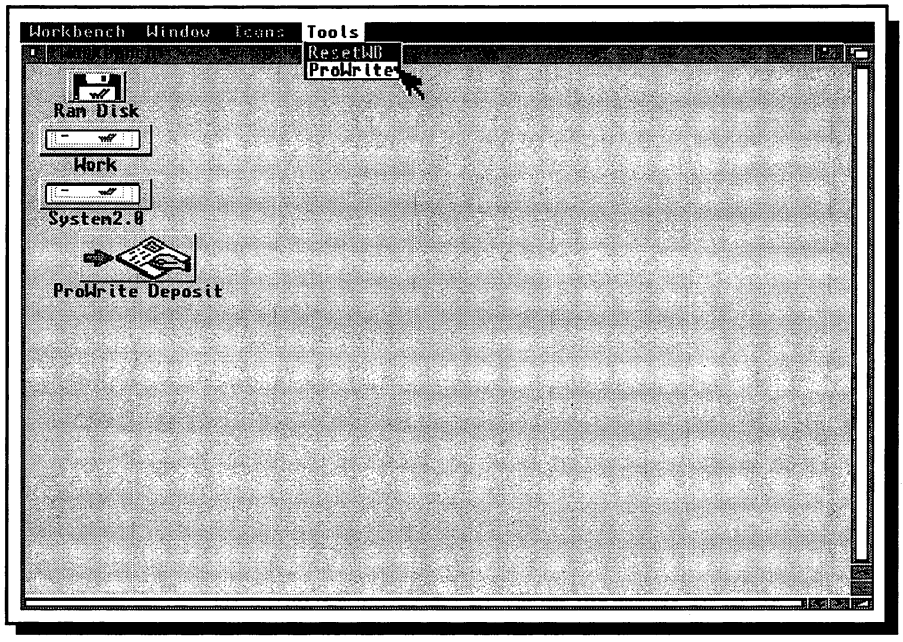
ResetWB: When invoked, ResetWB (which stands for reset Workbench) closes and then opens all Workbench windows, if possible. It does not work if nonWorkbench windows, such as a Shell, or applications windows, such as Clock, are open on the Workbench screen. If the close is successful, however, the Workbench reopens to reset any changes you've made in its colors, fonts, window patterns, or other Preferences items.

New to Workbench 2.0 is the capability — called AppMenuItem — of adding third-party applications programs to the Tools menu. For example, when you run version 3.1 of ProWrite, the program adds its name to the Tools menu (see Figure 3-17). Whenever you choose ProWrite from the menu, its screen is popped to the front of the display and its top window is made active. No action on your part is needed to install an application in the Tools menu; applications that support this feature provide it automatically.

Also new to Workbench 2.0 is another way to load projects into programs and to bring a program's screen to the front of the display called AppIcon. Although not strictly a part of the AppMenuItem system, applications that support one will often support the other.

Like AppMenuItem, an AppIcon is an icon created by a particular program that appears in the main Workbench window. Dragging a project icon to the AppIcon loads the project into the application. Double-clicking on the icon brings the application to the front of the display.

Few applications support AppMenuItem and AppIcon at the present time. Many more will as developers release Workbench 2.0 versions of their programs.



3-17 AppMenuItem

When launched, the ProWrite word processor adds its name to the Tools menu. Selecting the ProWrite item brings the program to the front of the display.

Conclusion

The Workbench menu system is both easy to use and powerful. Although at first you may want to avoid some of the more complicated items such as the Information item, you can still perform all the basic tasks you expect from an operating system with very little effort. As you become familiar with the Amiga environment and more confident in your abilities, you'll be ready to take on Tool Types and protection bits. The next chapters discuss Preferences and other tools that come with the Workbench interface.

— 4 —

Preferences

One of the most important features of the Workbench interface is that you can customize the ways it looks and operates. Making your preferences known is a function of the Preferences editor programs in the Prefs drawer. Although Preferences has been an integral part of Workbench since version 1.0, it has been completely reworked in version 2.0.

Preferences Editors

The 13 Preferences editors let you modify the look and operation of Workbench. Many of the editors deal with the operation of peripheral devices that you hook up to your Amiga, such as a printer, modem, or monitor. Others let you control the visible attributes of Workbench; its colors, the fonts it uses, and so on. Still others let you perform such specialized functions as setting the internal clock-calendar and choosing keyboard equivalents to some common mouse functions. The changes you make to Workbench using the Preferences editors can be either temporary — in effect until you reboot your Amiga — or lasting. In the latter case, the selections will be saved to disk and reloaded each time you boot.

To find and reload your preferences every time you start your Amiga, the editors must save them in a special place. If you open the Prefs drawer and choose Show All from the Window menu, you'll see a drawer named Env-Archive. Inside is another drawer called Sys, that holds the files containing your Preferences settings. For example, your color preferences are stored in the file named palette.ilbm, while the name of any printer you've chosen with the Printer editor is stored in the file printer.prefs. Note that if you haven't made any Preference choices yet, these files will not exist. When you boot your com-

puter, it looks for your Preference settings in the Env-Archive/Sys drawer. If you move, rename, or delete the Prefs, Env-Archive, or Sys drawer, the system will be unable to locate your settings. Until you have a good understanding of AmigaDOS, don't mess with the default location of the Preference files.

Hot Links

Your Preference settings are used not only at startup, but also throughout a computing session. For example, the WBPattern editor lets you choose one pattern for the background of your main Workbench window and another pattern for disk and drawer windows. The system reads your pattern settings at boot time. If you later change the pattern using the WBPattern editor, however, this system automatically communicates to Workbench, which institutes them immediately.

The new automatic communication facility used by the Preferences files and available to any Amiga program is officially called File Change Notification and unofficially called hot links. Any Amiga program can set up hot links to any file. Whenever such a file is changed, AmigaDOS sends a message to any program that has established a hot link to it. Thus when notified of a change, the program can take appropriate action.

In the case of window patterns, a special AmigaDOS program called IPrefs has hot links to both the wb.pat and the win.pat Preferences files. When you alter either file with the WBPattern editor, AmigaDOS informs IPrefs that something has happened to the files. IPrefs then checks the files and lets Workbench know about changes to the patterns it uses in its display.

The Two-File Solution

As I mentioned above, the Preferences editors let you make both changes that last only until you reboot, and more permanent ones that remain in effect after you reboot. Because both types involve writing to a file to trigger hot-link messages to Workbench, you may wonder how Workbench differentiates between them. The answer is that Workbench keeps two copies of each Preferences file; one for temporary changes and one for permanent ones.

The files for temporary Preferences changes are stored on the Ram Disk. If you open the Ram Disk and select the Show All menu item, you'll see three drawers: Env, T, and Clipboards. Inside the Env drawer is one named Sys. Inside this are the Preferences files. (T is a temporary storage area used by many programs; Clipboards is used by applications to cut and paste information be-

tween programs.) The similarity between the drawer names for temporary Preferences storage in the Ram Disk and the permanent Preference files in the Env-Archive/Sys drawer is not coincidental. When you boot your computer, AmigaDOS copies all the drawers and files contained in Env-Archive to the Env directory on the Ram Disk.

The window of each Preferences editor has three action buttons: Save, Use, and Cancel. Cancel, of course, exits the editor without making any changes. Use saves the changes you've made to the temporary file on the Ram Disk, which is the file with the hot link to Workbench via IPrefs. Save saves your changes to the Preferences files in both the Ram Disk and in the Env-Archive drawer. Thus, because saved changes are written to the Ram Disk file, which has a hot link to Workbench, and also to the disk file in Env-Archive, such changes both take effect right away and survive a system reboot.

The very first time you open a Preferences editor after installing Amiga OS 2.0, it will try to read the previous settings you saved to the Env drawer on the Ram Disk. It will, of course, fail because you have yet to save any settings. The editor will put up one or more requesters saying that it can't read its particular Preference file or files. Simply select the OK button when this happens. A new requester appears, stating that the editor will use the Workbench default settings. After you select OK, the editor window appears. Note that some editors such as the Font editor that create and use more than one Preference file will put up one requester for each settings file they work with.

The Preferences Menus

Except for the Time editor, the Preferences editors share a common set of menus that provide a way to save and recall alternate settings. In addition, most of the editors let you save these alternate settings with an icon. After exiting an editor, you can switch from your current setting to an alternate one simply by double-clicking on the icon of the alternate.

The Preferences menus are Project, Edit, and Options.

Project: The project menu has three items: Open, Save As, and Quit.

Open: The first item lets you open a file preference setting you previously saved to disk. Although you can save these settings anywhere, the default place to store them is the Presets drawer in the Prefs window. The files are called preset files.

The exact contents of a preset file depends on which editor you're using. If you are in the Fonts editor, for example, the file contains the name of one

of the three fonts this editor lets you set. If you're in the WBPattern editor, the file contains pattern information for the main Workbench window or for disk and drawer windows. Preset files don't contain the current Preferences settings, which are stored in the Ram Disk and in the Env-Archive drawer, but alternative settings that you may want to substitute for the current ones.

The Open item brings up the standard Amiga file requester (see Figure 4-1), which lets you load a preset file into the editor. A file requester consists of three parts: a file list, one or more text gadgets, and some action buttons. In the Open file requester, the file list contains the names of all the files in the current drawer in black type on a grey background. If the list display is too small to show all the files at once, the scroll gadget on the right lets you move through the list. Using the scroll gadget also alphabetizes the list and moves drawer names (indicated by the notation DRW) to the end. The name of the currently selected file appears in reverse (grey on black).

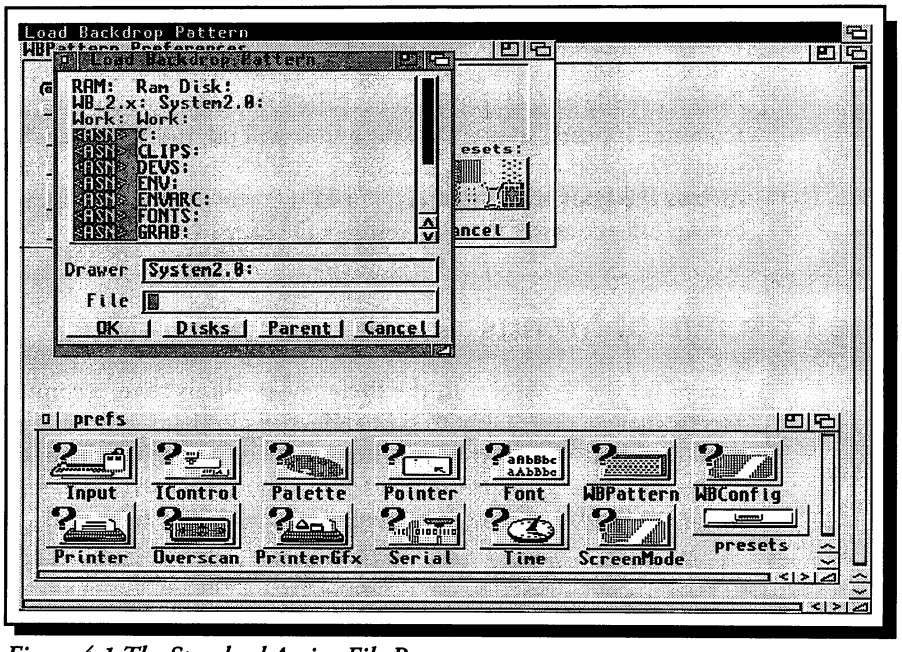


Figure 4-1 The Standard Amiga File Requester

With Amiga OS 2.0, Commodore finally supplies a standard file requester. Up to now, software companies have had to "roll their own" requesters, which didn't make it easy when you moved from one program to another. As more companies upgrade their products to 2.0, you will make greater use of the standard requester.

Below the list window are two text gadgets. The top one shows the current drawer, while the bottom one shows either the name of a default file or the name of a file you selected from the list. The current drawer is the drawer whose contents are listed in the file list. With the Preferences editors, the current drawer defaults to Sys:Prefs/Presets. This is AmigaDOS shorthand for the Presets drawer which is found in the Prefs drawer, located on the system (Sys:) disk. The Preferences editors also put a default filename in the File test gadget. For the Fonts editor, this filename is fonts.pre; for the WBConfig editor, it is wbconfig.pre; and so on.

At the bottom of the requester are buttons: OK loads the file named in the file gadget into the editor. Cancel aborts the Open procedure (as does the close gadget in the upper-left of the requester). The Disks button brings up a list of all the disks *and* all the assigned directories currently active on your system. (Assigned directories are special places that AmigaDOS can access without having to use a complete pathname. You'll learn more about them in the section on AmigaDOS.) Disks are listed in blue text first by their AmigaDOS names and then by their Workbench volume names. Assignment names are preceded by <ASN>. The Parent button moves from the current drawer to the one above it in the hierarchy. For example, the parent of the Presets drawer is the Prefs drawer, because Presets is located within Prefs. If your current drawer is a disk or volume name, the Parent button has no effect because these are at the top of the AmigaDOS hierarchical file structure.

To select a file to load, you click on its filename in the list window and select the OK button or simply double-click on the filename.

To change the current drawer, you either select the Parent button, which moves you up the directory tree, or the name of any drawer shown in the list window, which moves you down into that drawer. Using the requester you can move to any directory on any disk in the system. With the Preferences editors, however, it's best to store all the presets in the Presets drawer.

The final element of a standard Amiga file requester is an associated menu. Four of the items in the menu — OK, Disks, Parent, and Cancel — mimic the functions of the action buttons. They are included in the menu primarily to provide keyboard equivalents to the action buttons. Also included are two items that let you go through the files in the file list in alphabetical order: Last Name replaces the filename in the file gadget with the filename that alphabetically preceded it in the list of files. Next Name replaces the file with the one that follows it alphabetically. Both of these items have keyboard equivalents.

Save As: The second Project menu item lets you save a preset file. Save As also brings up a file requester that allows you to access any file on your system. The only difference is that the save requester lists text as gray on black.

Like the Open item, Save As puts a default drawer into the Drawer gadget and a default filename into the File gadget. Stick with the default drawer for storing presets. Use filenames that will make sense to you six months or more from now and that relate to the editor that created the file.

Quit: Finally, Quit lets you exit the editor. It functions the same as clicking on the Close gadget or the Cancel button.

Edit: The Edit menu has three permanent items: Reset to Defaults, Last Saved, and Restore. It may also contain other items, depending upon the editor.

Reset to Defaults: If you've overdone it in the customization department, this item restores the settings in an editor to those selected by Commodore and shipped on your Workbench 2.0 disk.

Last Saved: This item reads the settings you last saved to the Sys drawer within the Env-Archive drawer, not any you saved as a preset.

Restore: Restore returns the editor to the settings it had when you opened it. Any changes you've made are lost.

Optional items: The Edit menu can also contain items specific to the current editor. For example, the Edit menu of the WBPattern editor contains an Undo item, while the Edit menu of Palette contains nine built-in presets. Unlike the presets accessed through the Project menu and stored on disk, these are a permanent part of the Palette editor.

Options: The Options menu has one item, Save Icons?, which isn't available on all editors.

Save Icons? Many editors give you the option of saving your preset files with an icon. When you double-click this icon, it loads the editor from the disk and resets the temporary Preference file with the information from the preset. The editor then exits immediately, without displaying its window. Using presets with icons, you can switch between two or more Preference settings with the double-click of a button.

For example, let's assume that you used the Font editor to change the Workbench's default Topaz 8 font to some of the newer, better looking fonts available under Workbench 2.0. One problem you may encounter is that many programs written under earlier OS versions assume that Topaz 8 is and always will be the system default font. These programs may be unable to properly handle a larger font such as Helvetica 13 or Times 24. To get around this, you have to reset your font Preferences back to Topaz 8. The easiest way to do so is to create a preset file — perhaps called TopazBack.pre — and move it into the Prefs drawer. Now, before you run a program that expects Topaz 8, you simply double-click on the TopazBack.pre icon and it automatically and immediately resets your font selection.

All editors have the Preferences menus in common. Now, let's look at the individual editors and their unique functions.

The Font Editor

One of the nicest features of Workbench 2.0 is its support for multiple fonts. Previously, although individual programs on the Amiga supported multiple fonts, the Workbench display was stuck with one font, usually Topaz 8. The fonts editor makes it easy to replace Topaz 8 with up to three others.

The Font editor's window (see Figure 4-2) is divided into three areas. The top displays the three types of fonts you can have active on Workbench at one time and the current selections for each type. The buttons next to the text types let you choose which kind you want to modify: Workbench icon text, Screen text, and System default text.

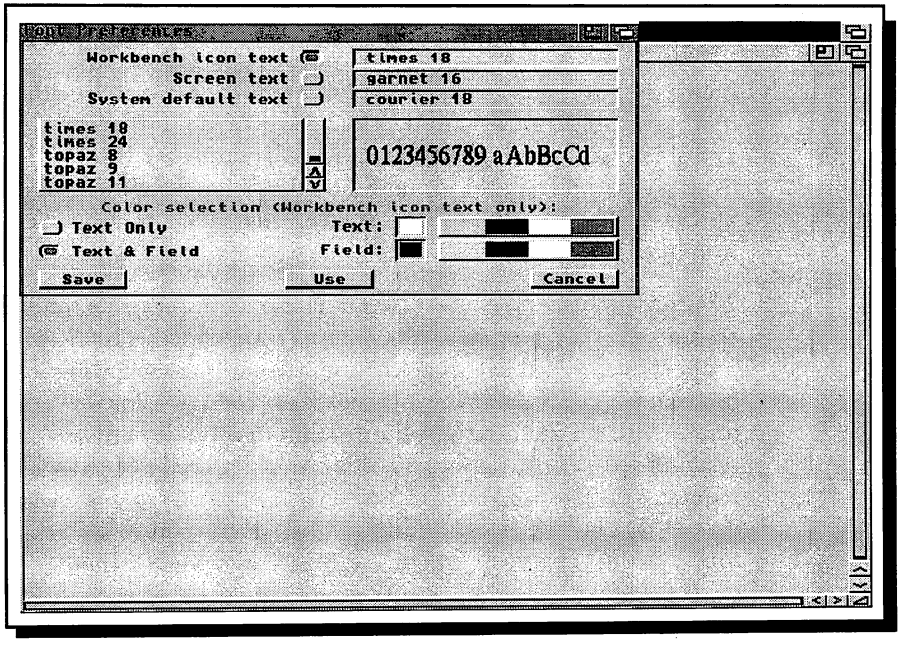


Figure 4-2 The Font Editor

The Font Editor lets you select different fonts for the three font types used by Workbench. The font list is context sensitive; it will only display proportional fonts when appropriate.

Workbench icon text: You select this button when you want to change the font, color, or background of the icon names on the Workbench display. Workbench icon text is the only type for which you can change both the text and background colors. (The text color and background settings — but not the font setting — are also used in windows where you choose one of the Workbench menu's View By options to display names instead of icons. In this case, the color and background settings apply to all text except the filenames.)

Screen text: This is the font that Workbench uses in its screen, window, and requester title bars, as well as its menus. If you substitute a larger font for Topaz 8, all your menus and title bars (and the gadgets within title bars) will grow vertically.

System default text: The font you select for the system default is used in the body of Shell windows and in windows for which you select a View By option other than View By Icons.

Note that some text you cannot change. The system uses Topaz 8 in the body of many requesters and windows, such as the Preferences editors' windows.

Below the text-type gadgets is the Font requester, which lists the fonts currently available for use with the selected text type. If the list is longer than the available space, you can use the scroll gadget to move through it. Next to the list is an example of the current font, whose name appears next to the active button in the text-type display. You can change the font by selecting a new one from the list.

The font requester is context sensitive; it only displays the fonts that are applicable to a certain text type. Amiga fonts come in two varieties, fixed length and proportional. Fixed length give each character the same width, even though, for example, an "m" obviously needs more room than an "i". Proportional fonts give each character just the amount of room it needs. Workbench icon text and screen text can handle proportional as well as fixed-width fonts, so the requester displays all your fonts when one of these types is selected. The system default text must be fixed-width, so the requester only displays fixed-width fonts for this text type.

If you're running Workbench 2.0 on a floppy-based system, you're probably wondering why you need a scrollable font requester to display three fonts, Topaz 8, Topaz 9, and Topaz 11. The other fonts are available on the Extras2.0 disk, but accessing and using them can be a real headache, especially if you have only one disk drive. See the section at the end of the chapter entitled Using Preferences with Floppy-Drive Systems for help.

Below the font requester is the color selector for the icon text and background. To the left are the mutually exclusive Text Only and Text & Field buttons. If

you want your icon text displayed on the normal window background, select Text Only. If you want icon text displayed on a rectangular background, select Text & Field.

To the right are the color gadgets for the icon text and background. Depending upon the number of Workbench colors you've set using the ScreenMode editor, you could have up to 16 colors displayed beside the the Text and Field buttons. The default is four. The colors themselves come from the current settings of the Palette editor.

To select a text color, you simply choose a color from the Text color gadget. It will immediately show up in the box next to the left of the gadget. You use the Field color gadget to select a background color. Note that Workbench won't keep you from your own folly. If you select identical text and background colors, that's what you'll get, even though you'll be unable to read the icon names. The same goes for selecting a text color in text-only mode that matches the background color of the Workbench windows.

Once you've made your font selections you use the action buttons to save, use, or cancel them. Figure 4-3 shows a Workbench screen that uses a white Times 18 font on a black field for the icon text, Garnet 16 for the screen text, and Courier 18 for the system default text. In addition to showing where the different text types are used, it is a good demonstration of how ugly things can get if your font selections get out of hand.

The IControl Editor

Workbench is based upon the underlying Intuition library of functions. The IControl Editor lets you change some of the control items the Intuition library makes available.

Figure 4-4 shows the window for the IControl editor and the five functions it controls: Verify Timeout, Command Keys, Mouse Screen Drag, Coercion, and Miscellaneous Flags. The IControl editor supports all the standard Preferences menu items.

Verify Timeout: On occasion, a program running under Workbench can get its communications with the Intuition library crossed up, usually when a program sets a MENUVERIFY flag. This flag tells Intuition to wait for confirmation that the program has completed a certain task before sending along a user's menu selection. If the program doesn't complete the task or fails to send confirmation, you get a deadlock with Intuition waiting to pass on a menu selection to a program not ready to receive it.

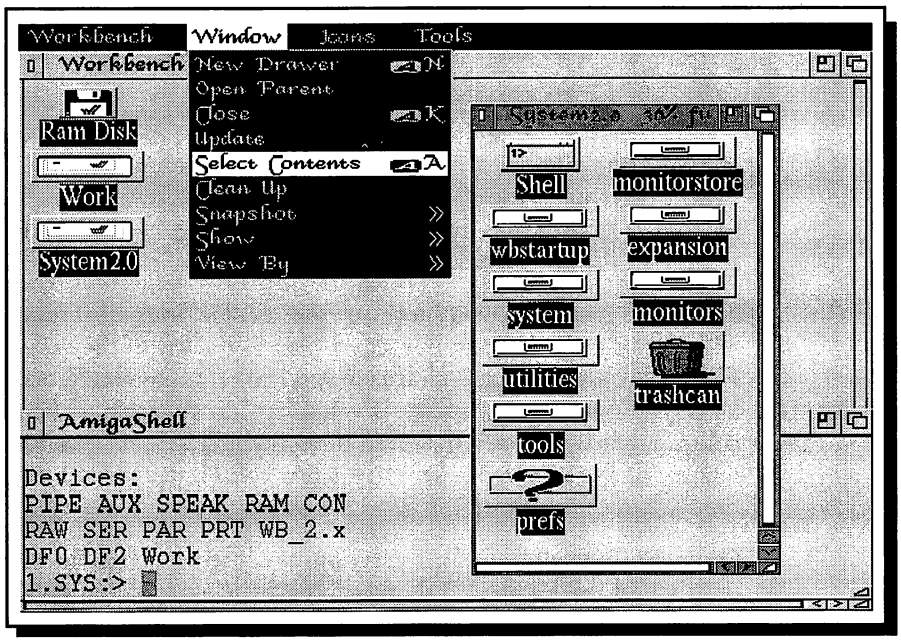


Figure 4-3 Effects of the Font Editor

This is certainly not an ideal use of the Font editor, but it lets you see exactly what text is controlled by what font type on the Workbench screen. (From this example, you can see that the Font editor obeys the most basic rule of computing: garbage in, garbage out.)

To break the deadlock, Intuition sets a time interval that it checks when it expects input from a program. If Intuition doesn't receive the message it expects within the interval, it proceeds to other things. The Verify Timeout gadget lets you set this interval.

Verify Timeout is an example of a mutual-exclusion gadget; select one of the buttons and all the others are automatically deselected. Commodore refers to such a gadget as a radio gadget, because the buttons act the same way as buttons on a car radio. You can set the timeout interval to one of five settings from 0.5 seconds to 5 seconds. The need for the interval is rare enough that I set mine at 5 seconds to give Intuition and the program as much time as possible to straighten out their communications.

Command Keys: While Workbench is primarily a mouse-driven interface, Intuition does provide for keyboard equivalents of some functions. Command Keys lets you set these equivalents using text gadgets. Because they mimic functions normally performed with the selection button, all use the Left-Amiga key.

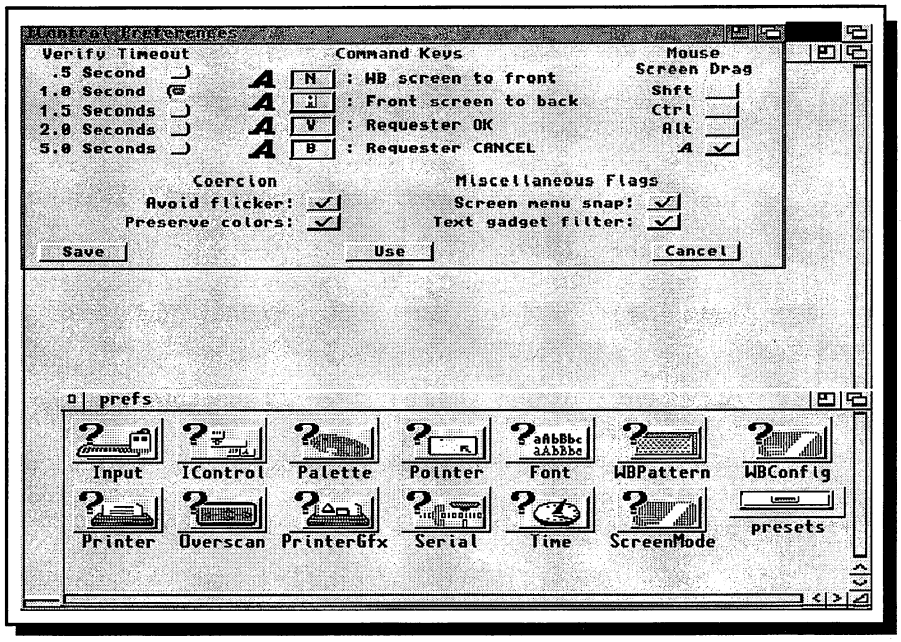


Figure 4-4 The IControl Editor

This editor gives you control over some of the functions of Intuition, the system of windows and menus upon which Workbench is built.

WB screen to front: Some programs that open custom screens don't provide a depth gadget or a drag bar to let you move from the custom screen to Workbench. In such situations, pressing the Left-Amiga key and N at the same time brings Workbench to the front of your display. You can change the N to another letter by selecting the text gadget and entering the new letter. Keep in mind, however, that some programs that close down the Workbench screen to save memory also disable the ability to move between screens using command keys.

Front screen to back: This command-key combination lets you flip through all the screens open on your system. It moves the front screen on your display to the back, thus revealing the screen underneath.

Requester OK: Pressing this command key combination is equivalent to selecting the OK, Retry, or Continue button on a Workbench requester with the mouse.

Requester Cancel: This command key combination is the equivalent of selecting the Cancel button on a Workbench requester.

After you change a command key in its requester, you must then press the Return key to record your choice. When you select the Use or Save buttons,

your choice will go into effect. The requester command keys work on requesters only, not on windows such as the IControl window.

Mouse Screen Drag: You can move any Amiga screen equipped with a drag bar up and down by dragging it with the selection button. If the screen is wider than your display (see the ScreenMode editor), you can also move it side to side. Mouse screen drag lets you turn any part of a screen into a drag bar.

By pressing one of the keys listed under Mouse Screen Drag and holding down the selection button, you can drag a screen. The selection button doesn't have to be over the drag bar; it can be on any window on a screen.

The Shift keys, Alt keys, Control, and Left-Amiga key are all available for the Screen Drag function. Unlike the Verify Timeout buttons, the Mouse Screen Drag buttons are not mutually exclusive, although they should be. Selecting more than one button turns off the Mouse Screen Drag function entirely, instead of making more than one key for the function, as you'd expect. Perhaps future releases of Workbench 2.0 will fix this "feature."

Coercion: The two buttons under this heading tell Intuition how to handle situations in which the screens on your system use different display modes. Intuition gives the topmost screen priority and displays it in its appropriate mode. If you drag that screen down part-way to reveal a screen that uses a different display mode, Intuition may have to "coerce" the underlying screen into a new mode if it uses one incompatible with the topmost screen. (If the underlying screen is using one of the A2024 modes, Intuition will simply make it invisible.)

This coercion sometimes results in severe aspect ration distortion of the underlying screen. To compensate, Intuition may switch the coerced screen to a faster pixel rate, thus sacrificing some colors, or make the coerced screen interlaced, to shorten the height of the pixels. You can tell Intuition which options your prefer with the Avoid Flicker and Preserve Colors buttons.

Avoid flicker: When selected, this button tells Intuition not to use interlacing. If you don't have a Display Enhancer (from Commodore) or a flickerFixer (from MicroWay) in your system, you should probably select this option.

Preserve Colors: Selecting this option tells Intuition not to switch to higher resolution pixels and drop some of the screen colors. A natural if you have a Display Enhancer or flickerFixer, this makes Intuition first resort to an interlaced screen to maintain the proper aspect ratio of the coerced screen.

Miscellaneous Flags: As the name implies, these buttons handle functions that didn't fit anywhere else.

Screen Menu Snap: Workbench 2.0 lets you create screens that are bigger than your output display and scroll around them using the drag bar, Mouse Screen Drag, or by moving the pointer to the border of the visible display. Because of this capability, the menu bar might not always be visible when you press the menu button. With the Screen Menu Snap feature selected, your display will always move to the upper-left corner of a screen when you hit the menu button. After you select an item, the display snaps back to your previous location on the screen. Disabling this feature means you will have to manually scroll the menu bar into view before you can use the menus. I can't imagine why you would want to turn Screen Menu Snap off.

Text Gadget Filter: With this option active, Intuition filters your keyboard input into a text gadget and removes non-printing characters created when you press the Control key in combination with an alphanumeric key. This keeps you from entering these invisible characters into filename requesters. When you begin using the Shell, you'll understand the problems nonprintable characters can create. To avoid that mess, keep this option active.

Even with the filter active, if necessary you can still enter control characters into text gadgets by pressing both the Control and the Left-Amiga keys in addition to the character key.

The Input Editor

For the most part, you use two devices to communicate with programs running on the Amiga: the keyboard and the mouse. The Input editor lets you change the characteristics of these devices to suit your tastes. With the Input Preferences requester (see Figure 4-5), you can set three mouse and two keyboard characteristics.

Mouse Speed: The slider at the top of the window lets you control how fast your pointer moves across the screen in response to mouse movement. By clicking in the slider box or dragging the slider, you can set mouse speed to 1 (the fastest), 2, or 4 (the slowest).

Mouse Speed actually tells Workbench how many pixels to move the pointer in response to the physical movement of the mouse. The fastest setting moves the pointer the greatest number of pixels per mouse movement, the slowest moves it the least number. The fastest setting is fine for everyday use and minimizes the amount of space your mouse needs to operate. The slower settings give you finer control over pointer positioning and are useful for detail work in painting or design programs.

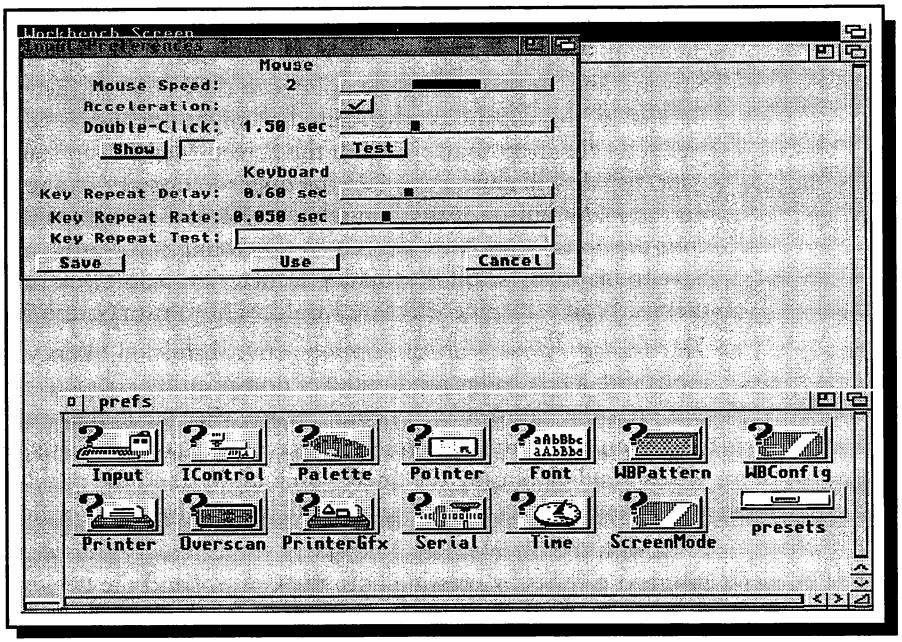


Figure 4-5 The Input Editor

This editor controls the characteristics of the two standard Amiga input devices, the mouse and the keyboard.

Acceleration: Below the mouse-speed slider is a check gadget that lets you turn mouse acceleration on and off. Click once and you turn acceleration on; click again and you turn it off. Acceleration controls how often Workbench draws the image of the pointer as you move the mouse. With acceleration on, Workbench reduces the number of times it draws the pointer, thus making the pointer appear to move faster. Acceleration also cuts down on the amount of room the mouse needs on your desk.

Double click: Workbench uses one click of the left mouse button to select objects and a double-click to open an object. It differentiates between a double-click and two single clicks by the amount of time between the clicks. The double-click slider lets you set the maximum time interval between the two clicks that constitute a double-click. If the interval is below the setting the two clicks are interpreted as one double-click; above the setting, and the two clicks are just that — two single clicks. Using the slider, you can adjust the interval from 0.2 seconds to four seconds. Clicking in the slider moves the bar .02 seconds in the direction of the pointer. You can also drag the bar to the desired setting.

The Input editor provides two gadgets that let you see the results of changing the double-click interval. Selecting the Show gadget paints a rectangle within the box to the gadget's right. The rectangle remains for the duration of the double-click interval, then disappears. The Test gadget lets you get the feel of the double-click interval. When you click twice on the gadget, the editor tells you whether the two clicks would constitute a double-click under the current interval setting. A yes means one double, no means two singles.

Key Repeat Delay: With the exception of special keys such as the Amiga, Control, Shift, Alt, and the function keys, the Amiga keyboard repeats if you hold down a key. Key repeat delay lets you specify how long you must hold a key before it repeats, from 0.2 seconds to 1.5 seconds. The slider is graduated in .02 second intervals.

Key Repeat Speed: This lets you set the interval between keystroke repeats, from as short as .002 seconds up to .250 seconds.

Key Repeat Test: With this gadget, you can test the settings you made with the delay and speed sliders. Click in the gadget, then hold down a key. The character you enter begins repeating after the interval you set with the delay slider and repeats the character at the rate you set with the speed slider.

The Input editor's settings are subjective, so I can't indicate which are best for you. If you're a heavy-handed typist, however, you should set a long repeat delay. If you're a bad typist, go for a fast repeat speed, which shortens the time needed to delete or backspace over your mistakes in a word processor. As with all the Preferences editors, selecting the Cancel button or the close gadget leaves the editor without making any changes. Selecting Use puts the changes into immediate effect without saving them to disk. If you reboot your machine, your original setting will again be in effect. Selecting Save both implements the changes immediately and alters your Preferences file so that the system uses the new settings the next time you boot.

The Overscan Editor

The display on your computer monitor consists of a rectangular array of pixels output by Workbench. This array does not quite fill your monitor screen, but you can enlarge its area using the Overscan editor. How large an area you can display depends upon your monitor type. The Amiga creates different sized overscan displays for different monitors. Before you use the Overscan editor, therefore, you must tell Workbench what type of monitor you are using.

You have three methods for indicating your monitor's type. The first is too do nothing. Amiga OS automatically defaults to an NTSC monitor type if you live in the United States and Canada and a PAL-type monitor for elsewhere. One of these monitor types will always appear in the Overscan editor window (see Figure 4-6). If you have a multiscan monitor or Commodore's A2024 monitor, you must take active measures to tell Workbench about them.

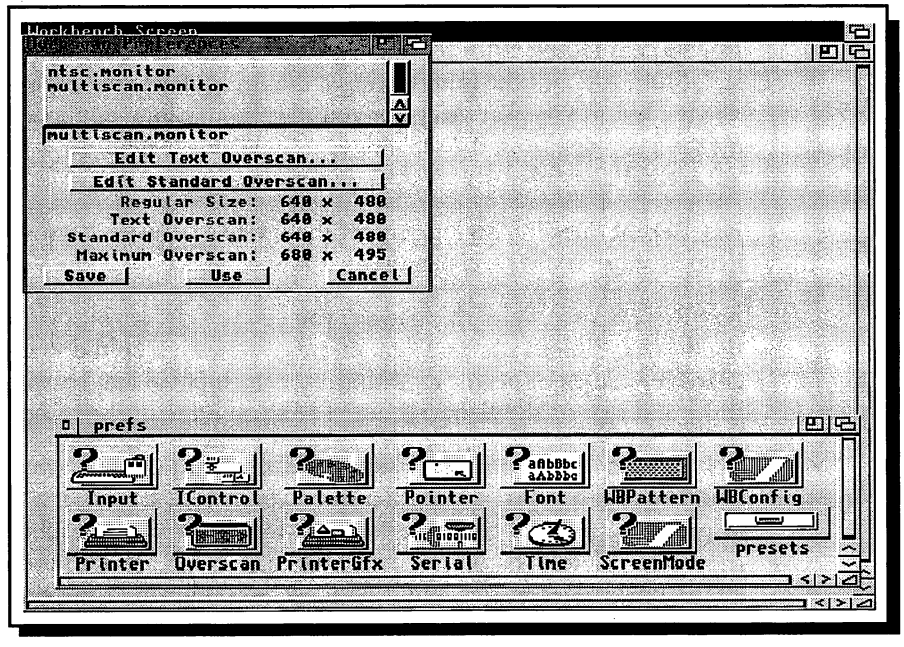


Figure 4-6 The Overscan Editor

The Overscan editor lets you select the monitor you're using and the type of overscan you wish to set: text or graphics.

Included in the Workbench 2.0 release is a drawer called MonitorStore, found on the System2.0 disk of hard-drive systems and on a floppy-only system's Extras2.0 disk. Inside this drawer are icons representing four types of monitors: NTSC, PAL, Multiscan, and the A2024. To indicate which you've attached to your system, simply drag its icon to the WBStartup drawer of your system disk. When you next boot your system, it will make the monitor type and its display modes available to you in both the Overscan editor and the ScreenMode editor.

My hard-drive system contains a third way to get the Workbench to recognize a monitor: the Monitor drawer on the system disk. Moving a monitor icon here also causes the system to recognize the monitor at startup.

You can, of course, lie to your system and indicate a monitor type that you don't have. This can lead to interesting results. For example, if you move the A2024 icon to the WBStartup drawer and then choose an A2024 mode from the ScreenMode editor, you'll get a wild display. Luckily, you will be able to see the icons well enough to go back into the ScreenMode editor and change to a normal display.

Once you've indicated your monitor type to Workbench, it will appear in the Overscan editor's selection box. You can indicate the dimension of the overscan screen you want by selecting the proper monitor from the box and accessing the two Overscan editing buttons.

Edit Text Overscan: Selecting this button lets you enlarge the area available to the Workbench interface. It brings up a display (see Figure 4-7) bordered by eight solid boxes and containing one in the middle. The thin line around the boxes corresponds to the current limit of the text display area. You change its size by dragging the border boxes towards the edge of the display and by dragging the middle box to center the overscan area. Pressing the Return key returns you to the Overscan editor with your setting intact; the Escape key returns without changes. The menu in the upper-left side of the display accomplishes the same thing.

Edit Standard Overscan: Working identically to Edit Text Overscan, Edit Standard Overscan enlarges the area in which programs can display graphics. The standard overscan area will always be as large as if not larger than the text overscan area. If you change the standard overscan area to be smaller than the text overscan, the editor automatically shrinks the text area to match the standard overscan.

Below these buttons is an information display detailing the pixel size of your currently selected overscan options. The first line lists the size of a standard display on the monitor type you've chosen. The second lists your current selection for the text overscan area's size, while the third lists your current settings for the standard overscan area. The fourth line lists the maximum display size of your monitor type's screen.

The larger display that comes from using overscan isn't free. You pay a performance penalty. If you're simply doing word processing or telecommunications, then the larger display is probably worth the performance penalty. With other, more computationally intensive tasks you should consider whether overscan is worth the price. Perhaps the most popular use for overscan is in video work. Previously, without overscan, Amiga graphics dumped to videotape had a black border around them. Amiga OS changes that by providing for transparent borders around genlocked screens, but overscan is still necessary if you want video titles and animation to fill the entire video frame.

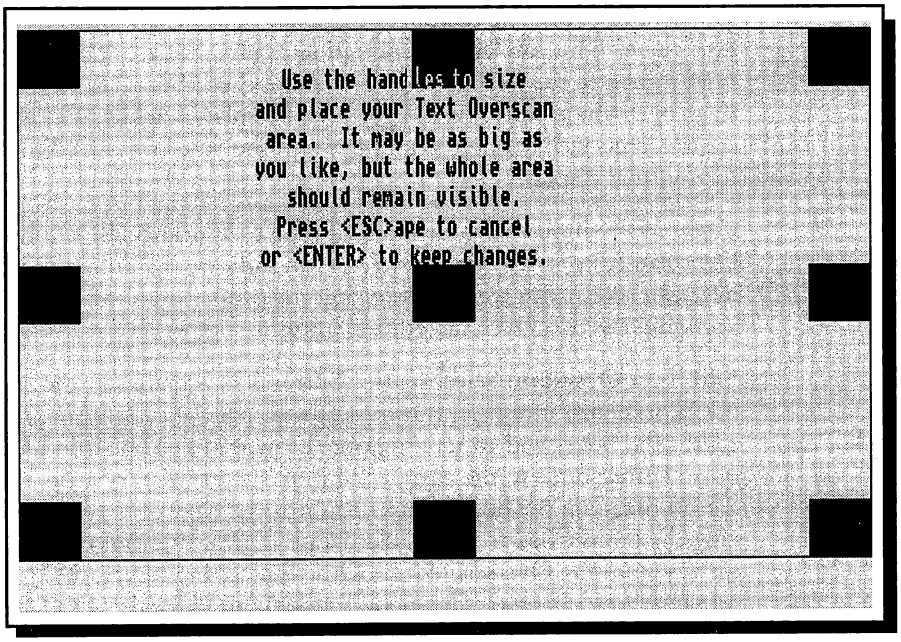


Figure 4-7 Edit Text Overscan

Text overscan is essentially the area where Workbench can render text. You set its dimensions by dragging the black handles with your mouse.

The Palette Editor

You can change the colors of your Workbench display with the Palette editor (Figure 4-8). The editor consists of a box that contains the current color you want to change, a color gadget for choosing the current color, and three sliders for changing the current color.

To edit a color, simply select one from the color gadget. This gadget displays from two to 16 colors, depending upon the number of colors you selected in the ScreenMode editor. When you select a color, it appears to the left of the color gadget and the sliders move to represent its red, green, and blue values. You change the color by moving the sliders.

Each slider can have one of 16 different values, from 0 to 15. A 0 represents the lack of a red-green-blue component from the final color; a 15 means that component is saturated. Thus, pure white is 15,15,15, while pure black is 0,0,0. These sliders let you create any one of the Amiga's 4,096 colors.

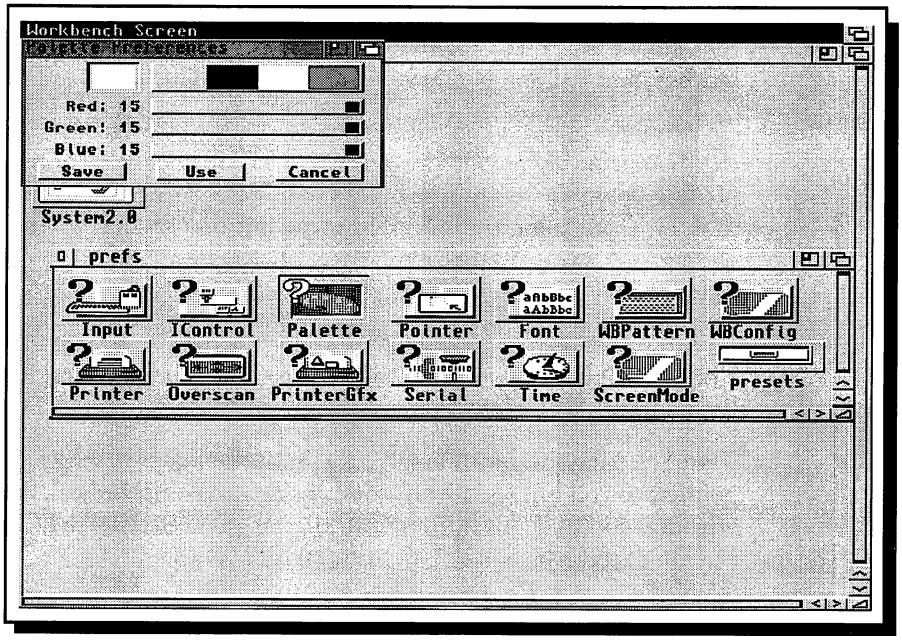


Figure 4-8 The Palette Editor

The Palette editor lets you select the Workbench colors. The color gadget automatically lists the number of colors you select in the ScreenMode editor.

The Palette editor doesn't let you save presets with icons, but it does have a number of built-in presets that you can try. They are listed as a submenu of the Edit menu's Presets item. All have keyboard equivalents that you can use while the Palette window is active. To make a preset permanent, choose it from the menu and click on the Save button.

The Pointer Editor

The Pointer editor is unique among the Preferences editors because it opens its own custom screen. This screen is low resolution — just 320 pixels across, instead of 640 — because the pointer is a low-resolution image. Also, Pointer needs more colors than the standard four-color Workbench provides because the pointer's colors don't come from the Workbench screen palette. You can confirm that Pointer opens its own screen by dragging the screen down with your mouse. Behind you'll find the Workbench screen.

The Pointer editor is a miniature paint program featuring an enlarged view of the current pointer, four views of the pointer showing how it looks superim-

posed over the Workbench screen's first four colors, a color gadget and slider that show the four colors presently used in the pointer, four special-purpose buttons, and the standard Save, Use, and Cancel buttons.

To edit your pointer, you select a color from the color gadget and begin painting. You paint by moving the pointer to the enlarged view of the pointer and holding down the left-mouse button. The pixels beneath the pointer will turn to the selected color until you release the button. To change colors, you simply select a new one from the color gadget. To create a new color, you move the sliders until you're satisfied. As you create or modify a pointer, check the four views to see how it will appear against Workbench's first four colors.

The four special buttons used with the Pointer editor are Test, Clear, Set Point, and Reset Color.

Test: Selecting this button makes the image you are working on the current pointer, so you can see how it works inside the Pointer editor. As a word of warning, don't hit this button immediately after selecting the Clear button. Nothing is more frustrating than an invisible pointer.

Clear: This button is fairly self-explanatory. It clears the enlarged image of the pointer so that you can create a new one from scratch. See the caution above about using this option with Test.

Set Point: Selecting Set Point tells the editor that the next point you select in the enlarged view of the pointer will be the active point. The active point is the part of the pointer that is used to select objects on the Workbench. As you can see, you can make fairly large pointers that can overlap multiple icons. The active point defines the true location of the pointer — the rest is just window dressing.

ResetColor: When you select this button, you return to the colors present when you opened the Pointer editor.

The Pointer editor supports the Save Icons? item in the Options menu as well as the standard Preferences menus. The Save, Use, and Cancel buttons perform their usual wizardry.

The Printer Editor

One of the more elegant aspects of Amiga OS is how it handles printers. Unlike the IBM world's MS-DOS, Amiga OS does not require that every application program provide built-in support for every type of printer in the world. Instead, all Amiga applications use a standard set of printer control codes. The

Amiga printer device takes these codes and translates them to printer-specific codes using the appropriate printer driver.

The Printer editor lets you pick your printer driver and set some printer default values. Workbench 2.0 ships with 25 printer drivers that, because of emulation of the most popular printers by other manufacturers, can support over 95% of the printers on the market.

If you don't find your printer on the list in the Printer editor, you will probably find a driver that your printer emulates. The documentation for Amiga OS 2.0 contains an extensive appendix on using various printers. If you can't find a driver that supports your printer, you can use the default generic driver for simple text output.

The Printer editor (see Figure 4-9) consists of a scrollable list of the printer drivers, three page-description text gadgets, and six buttons, as well as the usual three action buttons. It supports all the Preferences menu items.

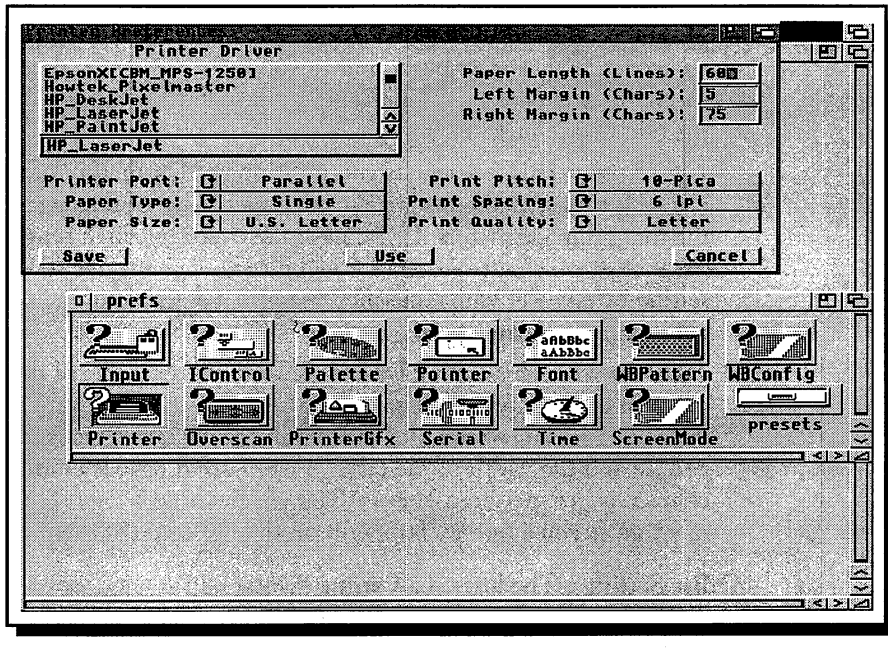


Figure 4-9 The Printer Editor

If the printer editor doesn't list your printer explicitly, you should try a compatible printer driver. If all else fails, you can select generic, which allows simple text output on nearly all printers.

Printer Driver: If you have a hard-drive equipped Amiga running Workbench 2.0, you'll see a scrollable list of the 25 printer drivers when you open the Printer editor. The small box below the list holds the name of the currently selected printer. If you've yet to select one, this box will contain the word generic. You select the driver for your printer with the left mouse button.

If you have a floppy-based system, however, your driver list will be blank; the Extras disk holds the drivers. To learn how to access these printer drivers skip ahead to the section entitled Using Preferences with Floppy-Drive Systems at the end of the chapter.

Page Settings: To the right of the printer drivers are three text gadgets that let you set the page length and margins of your printed output.

Paper Length: Use this gadget to enter the page length of the paper you're using. The default is 66 lines, which translates into an 11-inch page at 6 lines per inch. If you use some other length page or line-per-inch setting, you must alter this setting accordingly. Also, if the page breaks in your printed output aren't consistent, you may have to experiment with alternate page lengths. For example, I find the 60 lines per page is the proper setting for my Hewlett-Packard LaserJet even though I use letter-sized paper and the 6 lines-per-inch setting.

Left Margin: This setting lets you position the left margin of your printed pages. It measures the distance from the left edge of the page to the start of the margin in characters. If you have the character pitch set to 10 characters per inch, the default setting of 5 will give you a half-inch left margin. A setting of 10 gives you a one-inch margin. To get a one-inch margin at 15 cpi requires a setting of 15.

Right Margin: This setting is distance from the left edge of the page — given in the number of character positions — that you want the right margin to begin. The setting depends upon the character pitch setting you choose and the width of the page. A letter-sized page is 8.5 inches wide. Ten characters per inch translates into 85 character positions. The default setting of 75 gives you a one-inch margin on the right side of the page.

Other Settings: The lower half of the Printer editor window contains six buttons that let you set certain printer attributes. These buttons are called cycle gadgets, because whenever you select one, the text on the gadget changes to reflect another option. The active option is the one you leave showing when you exit the editor.

Printer Port: This button lets you cycle between parallel and serial. Choose the port to which you've attached your printer.

Paper Type: Here you indicate whether you are using fanfold paper or single sheets with your printer.

Paper Size: This gadget lets you cycle through seven different sizes of printer paper to choose the type you are using.

Print Pitch: Pitch is the number of characters per inch on a line of printed output. Your choices are 10, 12, and 15 characters per inch.

Print Spacing: This buttons lets you choose the number of printed lines that will appear in every inch of output. Your choices are 6 and 8 lines per inch.

Print Quality: With this you pick letter-quality mode or the coarser, yet faster, draft mode.

Note that many word processors and desktop publishing programs also let you determine these settings from within the program. In such cases the temporary setting from the program may override your Preferences settings.

The PrinterGfx Editor

While the Printer editor lets you choose your printer driver and set some basic attributes of the output page, it doesn't deal with the many variables necessary for the graphics output. Setting those options is the function of the PrinterGfx (the Gfx means graphics) editor.

The PrinterGfx editor has a lot of options, many of which require fairly technical explanations. You should read the explanations, both here and in the Commodore documentation, but the best way to learn about what this editor does it to *use* it.

When you open the PrinterGfx editor, you will see a window chock full of unfamiliar options (see Figure 4-10). Lets look at the easiest first.

Color Correct: Your Amiga can display up to 4,096 colors in HAM mode. When you print graphics to a color printer, however, the colors on the page don't always resemble those on your monitor.

The color-correct buttons let you select corrections for the red, green, and blue components of the colors in the printout. Selecting these buttons produce truer output, but will reduce the maximum number of colors in the printout by 308 for every button you select.

Smoothing: Next to the color-correction buttons is the smoothing button. With this button selected, the printer driver tries to smooth out diagonal lines that traditionally suffer from the jaggies. Smoothing results in better-looking output, but it can slow your printing speed to a crawl. Smoothing cannot be used with Floyd-Steinberg dithering.

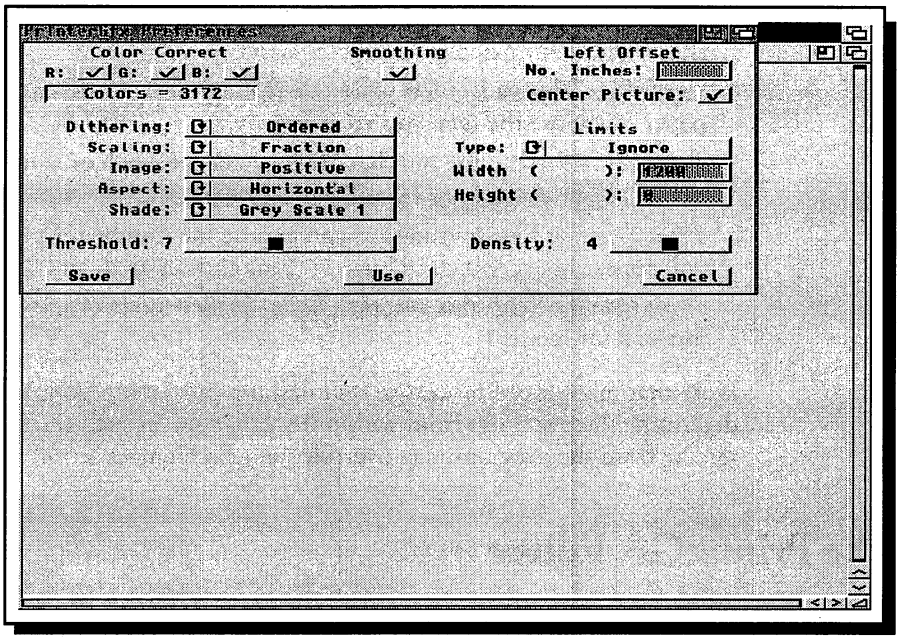


Figure 4-10 The PrinterGfx Editor

One of the more complex editors, PrinterGfx gives you a great deal of control over how your Amiga prints graphics. Remember, however, that as you increase the quality of your printouts, you also increase the time it takes to produce them.

Left Offset: Below the Left Offset heading are two gadgets, No. Inches and Center Picture.

No. Inches: You use this string gadget to enter the width of the left margin when printing graphics. You can enter the information in 0.1-inch increments.

Center Picture: Selecting this button centers your graphics output on the page and overrides the setting in the No. Inches gadget.

Dithering: Below the color-correction buttons are five buttons; the first, Dithering, cycles between three different dithering modes. Dithering is the process by which a printer combines dots of different colors to produce patterns that look like other colors. Most color printers only use four-color ribbons or four-ink jets. To produce thousands of colors requires that the printer mix dots of these primary colors so that the resultant patterns look like new colors. With grey-scale printing on a black-and-white printer, dithering varies the density of black dots on the white page. Greater density of dots means a darker shade, lesser density means a lighter one.

Ordered: This dithering method creates different colors by printing orderly rows and columns of colored dots. The dots have the same size and density. It produces grey scales by varying the pattern of the dots.

Halftone: Similar to the way newspapers print pictures, this method produces colors and grey scales by varying the size and density of the dots in a region.

Floyd-Steinberg: A complex mathematical dithering method, Floyd-Steinberg can produce excellent results. However, it is the slowest of the dithering methods.

Scaling: The next gadget controls the technique used to scale the output image. (You set the actual size of the image with the Limits button.)

Fraction: This option multiplies the horizontal and vertical dimensions of the image by the same factor to produce the desired image size.

Integer: Under the Integer option, each pixel in the image being printed will be represented by an integer multiple of dots on the output. This setting overrides the Aspect option setting and is useful for printing graphical text and images containing horizontal and vertical lines.

Image: The third button lets you select whether the image is printed as a *positive* image (what is black on the screen prints black on the paper) or as a *negative* image (what is black on the screen appears white on the printout).

Aspect: Using Aspect, you can choose between printing a *horizontal* image (oriented as its is on the screen) or a *vertical* image (prints sideways).

Shade: This button lets you indicate the colors or grey scales used to print your graphics.

Black & White: Selecting this option prints your graphics in black and white, without dithering or grey scales. Black & White uses the Threshold slider to determine which colors on the screen are printed as black and which as white.

Grey Scale1: To print colors as shades of grey, select Grey Scale1, which is the standard setting for black-and-white printers.

Grey Scale2: Designed to be used with the A2024 monitor, Grey Scale2 prints a maximum of four shades of grey.

Color: If you have a color printer, this setting will print your pictures in color.

Threshold: This setting applies only if you've selected Black & White as your Shade setting. It determines the dividing line between which on-screen colors print as black and which remain white. The lower the threshold, the fewer the on-screen colors that are printed black. Changing the Image setting from positive to negative means that values below the threshold will appear white on the printed output.

Type: To the right of the cycle gadgets is the Limits gadget, which lets you specify the size of your output. With it, you enter the *width* and *height* values for the printout into appropriate gadgets. How these values are interpreted, however, depends upon the selection you make with the Type button.

Ignore: As its name implies, Ignore ignores the values you place into the Height and Width text gadgets. It lets the application determine the size of the printout instead, considering the size of the paper and the margins set in the Printer editor.

Bounded: With this option the width and height settings are interpreted as tenths of an inch. Thus, a value of 50 equals five inches. Bounded sets and upper limit on the horizontal and vertical dimensions of the output image.

Absolute: This function interprets the Width and Height settings as absolute values. If you enter 60 and 50 into the Width and Height gadgets, respectively, you'll get a printout 6 inches wide and 5 inches tall. This could produce a severe aspect-ratio problem. To produce images with the proper aspect ratio, enter an absolute value into one gadget and enter 0 into the other. If you set both the Width and Height to 0, the output will be as wide as possible given the paper size and the margin settings and the height will be enough to ensure a proper aspect ratio.

Pixels: With Pixels, the Width and Height settings are interpreted as the number of dots used in each dimension. Entering 0 into one of the gadgets ensures a proper aspect ratio.

Multiply: Multiply treats the Width and Height values as multipliers. The system multiplies them by the number of pixels in the corresponding dimension of the screen image to determine the number of dots printed on the page. With one of the values set to 0, this option retains the aspect ratio of the screen image.

Density: A slider, Density determines the number of dots per inch used to print your images. A value of 1 corresponds to the lowest density output of which your printer is capable; a 7 corresponds to your printer's highest density level. Most printers can handle four or five levels. Check you printer's manual for details. Setting the density value above your printer's highest level ensures its highest density output. The higher the density, the better looking the output, but the longer it takes to print.

The ScreenMode Editor

One of the more important of the Preferences editors, ScreenMode lets you set your Workbench screen's display mode.

As I mentioned in Chapter 2, Amiga windows derive such basic properties as the number of colors they support and the maximum resolution they can attain from their underlying screen. Workbench windows are no different: They get their basic characteristics from the Workbench screen. The ScreenMode editor lets you alter those basic characteristics.

Choose Display Mode: At the upper-left of the ScreenMode window (see Figure 4-11) you'll find a scrollable list of the available display modes. The editor determines which modes to display here based upon the monitors you added to your system (either by placing them into the WBStartup drawer or by running the AddMonitor program). In addition, the editor always displays the NTSC or PAL modes, because, subject to restrictions, these modes are always available to you. The currently selected mode appears below the list display.

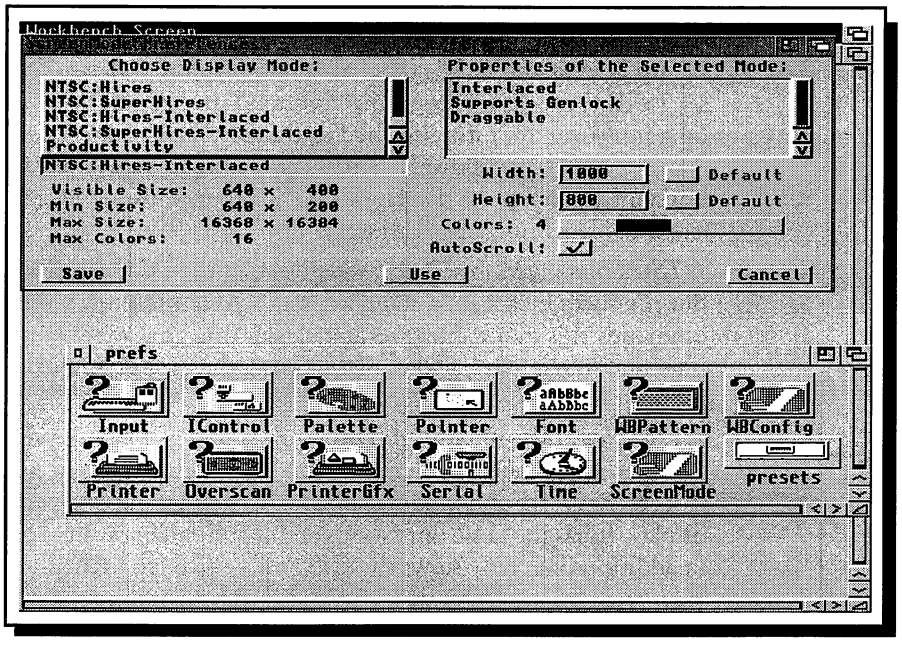


Figure 4-11 The ScreenMode Editor

This editor lets you set the basic characteristics of the Workbench screen, and, for the first time, permits a screen with more than four colors. With Commodore rumored to be working on an 8-bit display chip set due in early '91, ScreenMode may soon let you create a 256-color Workbench.

Note that the display modes listed are not the only ones available on your Amiga, but the only ones available to Workbench. Many programs open cus-

tom screens that use one of the other Amiga display modes such as lo-res, lo-res interlaced, and the HAM and Extra_Halfbrite (EHB) modes.

If you loaded up all the monitor types into the WBStartup drawer and booted your system, you'd discover that Workbench supports the following display modes:

NTSC Hires: Hires stands for high-resolution. Any Amiga screen whose visible area is 640 pixels wide is called hi-res. The NTSC designation means that the standard number of vertical pixels is 200. Like all Amiga hi-res modes, it lets you use up to 16 colors (out of a possible 4,096) at once. For Amiga 500 owners, or for Amiga 2000 owners who don't own either a Display Enhancer or a flickerFixer, this will probably be the most useful display mode because the screen doesn't flicker.

NTSC SuperHires: This mode features a horizontal resolution of 1,280 pixels although it limits you to four colors. Requiring the Enhanced Chip Set be installed in your Amiga, NTSC SuperHires is designed for use with a 15.75 KHz monitor such as the Commodore 1084. If you view its output on a multiscan monitor attached to the Display Enhancer circuitry on the Amiga 3000 (or on a display enhancer card in an Amiga 2000), every other horizontal pixel in the display will be missing. This mode is designed primarily for video-titling applications.

NTSC Hires-Interlaced: This mode doubles the number of horizontal pixels but produces flicker on a standard 15.75 KHz monitor. Commodore recommends this mode, however, for those Amigas that contain hardware intended to eliminate flicker by buffering the output and using a 31.5 KHz monitor, such as a multiscan.

NTSC SuperHires-Interlaced: This mode produces twice the vertical resolution of SuperHires. Viewed on a multiscan through the Display Enhancer circuitry, it loses every other horizontal pixel and also produces an unsteady looking display. It has the same limitations and requirements as the SuperHires mode; in addition, it greatly slows the CPU's access to chip RAM.

PAL Hires: Features the European television standard 256-pixel vertical resolution instead of the North American 200-pixel vertical resolution. You can view this mode through an NTSC or multiscan monitor, but the 50 Hz PAL refresh rate produces an unsettling display.

PAL SuperHires: Once again, this mode is the same as its NTSC cousin except that the vertical resolution is 256 pixels.

PAL Hires-Interlaced: Here the vertical resolution is 512 pixels. The mode requires a Display Enhancer or a flickerFixer to eliminate interlace flicker and looks strange on a 60 Hz NTSC monitor.

PAL SuperHires-Interlaced: Choose this mode for a 1,280-by-512-pixel display. You'll get the same limitations and requirements, however, as the NTSC version.

Productivity: Even if you don't have specialized hardware, Productivity mode produces a four-color, 640-by-480 pixel display that doesn't flicker. You will need the ECS and a multiscan monitor. Commodore designed Productivity mode for people who wanted more vertical resolution but who didn't have access to flicker-elimination boards. Like SuperHires, this mode ties up chip RAM a great deal, shutting out the CPU from accessing it. Called bus contentions, this condition can degrade system performance significantly. You may want to limit your use of productivity mode to such noncomputationally intensive applications as word processing. (Note: On an Amiga 3000, the bus contention is significantly reduced because of the 32-bit access the CPU has to chip RAM.)

Productivity-Interlaced: Would you believe Commodore added a 640-by-960-pixel display that flickers even on a multiscan monitor attached to a Display Enhancer or flickerFixer? To get a steady display with this mode would require a 65 or a 70 KHz monitor. Then there's the bus contention to worry about.

A2024_10Hz: This mode produces a 1,008-by-800, four grey-scale display on a Commodore A2024 display. The monitor builds it display from separate "panels" output from the Amiga. Watching this display on a normal monitor is an experience to be missed.

A2024_15Hz: Producing the same display resolution as the 10 Hz mode, A2024_15Hz updates the screen more frequently for a steadier display. The trade off is that this mode consumes more processing power from the computer than does the 10 Hz mode.

Properties of the selected mode: To the right of the mode display list is another display box. When you select a display mode, this box automatically lists some of that mode's properties.

Supports Genlock: A genlock synchronizes the Amiga output with an external video signal, allowing you to overlay graphics on a video picture. This mode supports genlocks by letting you designate any of its colors as the ChromaKey, the color that is replaced by the video signal in the combined picture. It also lets any bitplane enable the video display and can provide either a transparent or opaque border around the screen.

Draggable: These modes support screens that can be pulled down to reveal the screens behind them.

ECS: To use these modes, your computer must be equipped with the Enhanced Chip Set.

Requires bypassing the Display Enhancer: Obviously, these modes don't work well with the Display Enhancer.

Interlaced: The down side is that this produces a flicker on 15.75 KHz monitors; on the other hand, the display's vertical resolution doubles.

PAL: If you have the ECS installed on your NTSC machines, you can use modes with this property.

NTSC: The mode is available to PAL machines with the ECS installed.

Panelled: On monitors like the A2024, Panelled indicates the use of combined display panels to produce output.

Does not support Genlock, Not draggable: Need I say more?

Screen Sizes: Whenever you select a display mode, information about its size characteristics appears below the list of modes.

Visible Size: Check here for the dimensions of a screen in pixels when it fills your output display.

Min Size: Screens can be shorter than the output display. This item lists the smallest vertical resolution.

Max Size: One of the exciting things about Workbench 2.0 is that screens can be bigger — a lot bigger — than their visible size. When a screen takes up more than the visible display, you can scroll it using the mouse.

Max Colors: The maximum number of colors the mode can display at once is shown here.

Under the Properties list are a couple of text gadgets that let you set the size of a screen, a slider for selecting the number of colors in the screen, and a check gadget.

Width: Width lets you enter the width of the screen, up to the maximum width shown in the Max Size line. Selecting the default sets this value to the horizontal dimension of the Visible Size.

Height: You use this text gadget to set the maximum height of the screen, up to the value shown in the Max Size display. Selecting the default button sets this value to the vertical dimension of the Visible Size. Note that the size of your screen may also be limited by the availability of chip RAM.

Colors: Four is the default number of colors for a Workbench screen, although many modes allow up to 16. This slider lets you choose the number of colors supported by your Workbench screen. Note that 16 colors with a Hires-Interlaced screen will produce considerable bus contention on all Amigas except the Amiga 3000. More colors also mean greater memory requirements. For a 640-

by-400 pixel screen, you increase memory usage by 32K every time you double the number of colors.

AutoScroll: When a screen is bigger than its visible size, you need a way to move around to see it all. With this check gadget selected, the display scrolls when you move the pointer to an edge to reveal new portions of the screen. Without AutoScroll selected, you have to use the drag bar or mouse screen drag to scroll around the screen.

At the bottom of the window are the familiar Save, Use, and Cancel buttons.

ScreenMode is an important editor, one that you must understand well to make intelligent selections. I recommend that you start out conservatively by limiting your Workbench screen colors to four and your screen to the visible size. You can always experiment when you have more confidence and experience.

The Serial Editor

Many printers and all modems exchange data with your Amiga using serial communications. In serial communications, one device sends data as a stream of bits to another. To work, both devices must be packaging the bits in the same way and be transmitting and receiving them at the same rate. They must also establish a system so that when one is ready to send data, the other is ready to receive. The items that define such a system collectively are called the serial communications parameters.

The Serial editor (see Figure 4-12) lets you set the communications parameters for your Amiga's serial port. For the most part, you'll use this editor if you have a serial printer. Check your printer documentation for the proper parameters and set the serial port to match them. When using a modem, you'll discover that your telecommunications program lets you set the communications parameters from inside the program.

Baud Rate: The slider at the top of the window lets you set the baud rate, which is effectively the number of bits the port will send or receive per second. To communicate, two devices must use the same baud rate. Most printers work at 9,600 baud, most modems at 2,400. AmigaDOS 2.0 supports baud rates up to 31,250.

Input Buffer: Below the Baud Rate slider is one that lets you set the size of the input buffer for the serial port. 512 bytes is sufficient for most applications. If your Amiga holds up the device on the other end of the serial connection, however, you can increase this setting up to 64K bytes.

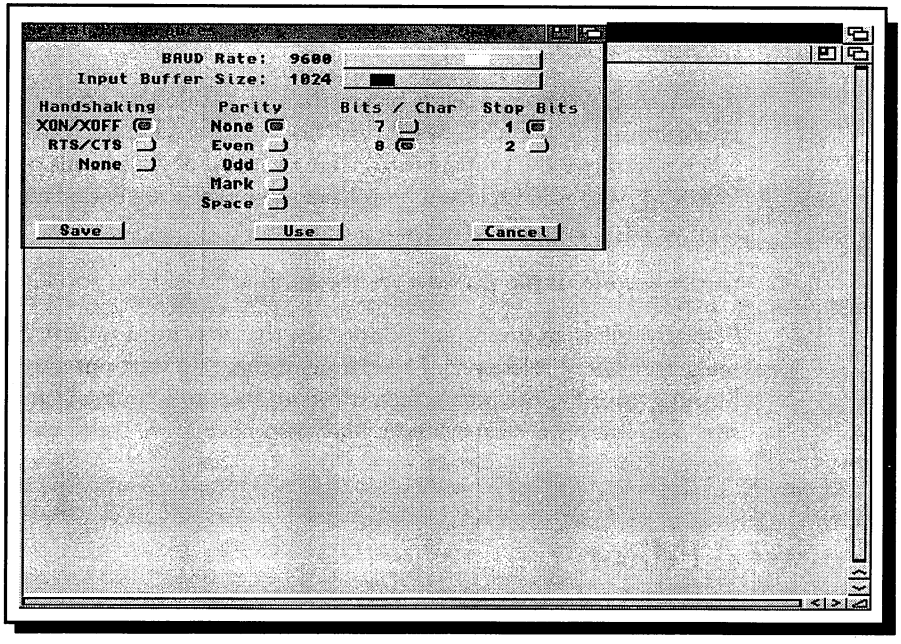


Figure 4-12 The Serial Editor

Because telecommunications packages usually set the serial port themselves, you'll only need to access this editor if you use a serial printer or make a direct connection with another computer.

Handshaking: Serial devices must establish a protocol that keeps both devices from sending data at the same time. You select a particular handshaking method by clicking the button next to your choice. Two types of serial protocols exist, software and hardware.

xON/xOFF: The software communications protocol xON/xOFF is the most common serial communications protocol. Under it, the different devices send special control characters to one another to indicate when they can and cannot receive data.

RTS/CTS: Used mostly for communications with printers, the protocol uses two physical lines within the serial cable to signal a request-to-send or a clear-to-send signal.

None: This option turns off all serial handshaking. It may be handy when you're simply dumping data from one device to another, but I've never used it and can't image ever doing so.

Note that these handshaking protocols are not the same as the binary transfer protocols, such as Xmodem and Kermit, that you use in telecommunications.

Such upper-level protocols are concerned more with high-level error checking than with basic handshaking.

Parity: Parity checking is a simple error-checking protocol. It helps ensure that data bits have been transmitted without error. In serial communications, data bits are sent in bunches — called words — of either seven or eight bits. When seven bits are used, the system can employ the eighth bit for parity checking. The radio buttons below the Parity heading list the parity options.

None: Under this option, data bits are sent eight bits at a time; no parity checking is done. Most telecommunications networks use the None parity setting.

Even: As the name implies, with even parity the number of 1 bits in the data word must be an even number. If the number of 1 bits is odd, the sender attaches a 1 parity bit. If the number is even, a 0 is attached. If the receiver detects an odd number of 1 bits, it knows an error has occurred.

Odd: In this case, the number of 1 bits sent is always odd. The sender adds a 0 or a 1 to fulfill this condition.

Mark: With this setting, the eighth bit is always a 1.

Space: Select Space and the eighth bit will always be a 0.

Bits/Char: This radio button lets you specify the number of data bits in a word. If you're not using parity checking, you should set Bits/Char to 8. Using parity checking, you should use 7 bits per character.

Stop Bits: These bits provide a timeout so that the receiving computer can digest the previously sent data bits. Your choices are 1 or 2 stop bits. The 1 setting is used almost universally today.

The Time Editor

The Time editor sets your Amiga's system clock and its battery-backed clock. The system clock is read by the Clock program and by AmigaDOS when it puts a creation date and time on a file. All Amigas come with a system clock. With the exception of the original Amiga 1000 and a stock 512K Amiga 500, all Amigas also come equipped with a battery-backed clock. Unlike the system clock, this one doesn't lose its information when your computer loses power. The system uses the battery-backed clock to reset the system clock when you boot your Amiga.

The Time editor (see Figure 4-13) lets you set both clocks' time and calendar information.

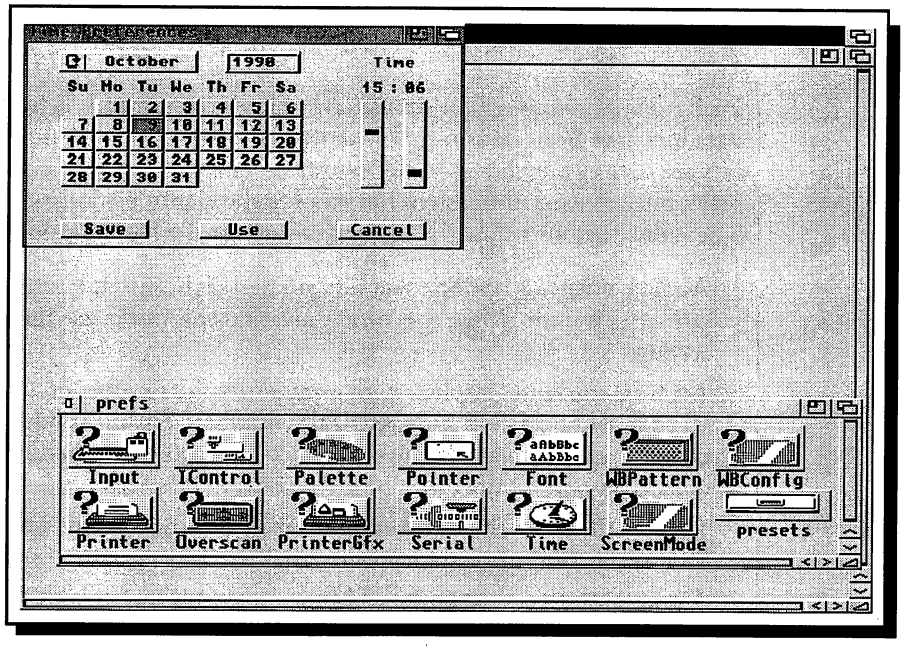


Figure 4-13 The Time Editor

You'll find that setting the time with this editor is much simpler than using the AmigaDOS Date command.

Calendar: When you open the editor, you should first enter the proper year in the text gadget at the top-center of the window. (Be sure to press Return!) Next, use the cycle gadget to the left to choose the month. The editor contains a perpetual calendar, so after you enter the month and year, the proper calendar will appear below. To set the day of the month, simply select it with the left mouse button.

Time: You set the time of day with the two sliders at the right side of the window. The left slider sets the hour; the right one sets the minute. Because the hour slider does not recognize AM and PM, you choose values from 0 (12 AM) to 23 (11 PM). The minute slider lets you select a value from 0 to 59.

The Use button sets the system clock, and the Save button sets both the system and the battery-backed clocks. If you have a battery-backed clock, you should have to use this editor only when you first unpack your new Amiga, move to a new time zone, or daylight savings begins and ends.

The WBConfig Editor

The simplest of the Preferences editors, the WBConfig editor contains two check buttons.

Workbench window as backdrop: As if the Workbench menu Backdrop item wasn't enough, Commodore presents this button. By selecting it, you can choose to have the main Workbench window appear as a standard resizable, draggable, zoomable window or as a backdrop window. Save your selection to have Workbench boot to the new specifications.

Double click for window to front: When I showed the Workbench 2.0 interface to a Macintosh aficionado, he commented that the interface was much improved but that he still didn't like using depth gadgets to bring windows to the front. (On the Mac, clicking in a window automatically brings it to the front.) I agreed, but I pointed out that automatically bringing a window to the front can be just as annoying. Imagine my delight when I later discovered that, on the Amiga, you can have it both ways.

With this option selected, you can bring a window to the front of the screen by double-clicking in the background area of the window. Double-clicking on an icon works the same as before, while double-clicking on the borders has no effect. I strongly recommend that you use this option.

The WBPattern Editor

Tired of that drab background you find in every Workbench window? The WBPattern editor can liven up your computer display. The WBPattern editor lets you select new background patterns for both the main Workbench window and the windows associated with disks and drawers. You can either create your own patterns or choose one of the editor's presets.

Pattern: When you open the WBPattern editor, you see the display shown in Figure 4-14. The radio button at the top left switches between Workbench and Windows, letting you select which type of window pattern you want to work on. You can have different patterns for each type of window.

Views: To the right of the Pattern button is an enlarged view box that resembles a miniature paint program. By holding down the selection button, you create a blown-up version of your pattern. The second view area in the upper-right of the window shows how the pattern in progress will look at normal size. It shows 12 iterations. When you use a pattern in a real window, it repeats until it fills the window.

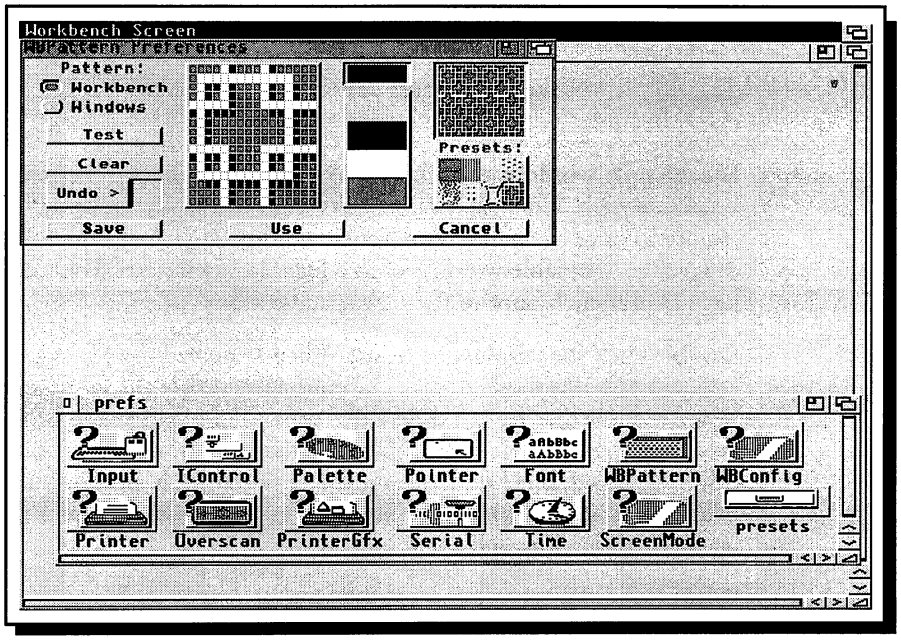


Figure 4-14 The WBPattern Editor

As with the Palette editor, the number of colors available with the WBPattern editor is determined by the ScreenMode editor.

Color Gadget: To the right of the enlarged view is a color gadget. The number of colors it displays depends upon the number you selected in the ScreenMode editor; the colors themselves depend upon the choices you've made with the Palette editor. At the top of the color buttons is a box showing the currently selected color.

Presets: Below the normal-sized view area are eight pattern presets. When you select one of these, the pattern immediately appears in the enlarged view and normal-sized view areas, overwriting what was there. You can then modify the preset to your heart's content. (Actually, the original preset remains unchanged; you are modifying a copy.)

Buttons: Finally, below the Pattern buttons are three special buttons. These are Test, Clear, and Undo.

Test: If you're not satisfied with the limited number of iterations provided in the normal-sized viewing area, you can test your pattern by selecting this button. Test uses the pattern in the enlarged area on the window-type specified by the Pattern buttons. Unlike the view areas in the editor, however,

Test does not automatically display subsequent pattern modifications. If you exit the editor without saving or using the last pattern tested, the target window type will revert to its original appearance.

Clear: Selecting this button paints the entire enlarged view area with the color shown at the top of the color gadget.

Undo: Whenever you release the selection button after making a change in the enlarged view, the editor records the pattern in temporary memory. When you next make changes with the selection button, the editor shows the enlarged view's previous state in the small box to the right of the Undo button. Selecting Undo restores the pattern in this box to the enlarged view. Undo is also available as an item in the Edit menu.

Using Preferences with Floppy-Drive Systems

If your Amiga is equipped with floppy drives only, you will run into problems using some of the Preferences editors — the Font and Printer editors in particular. These programs expect the font and printer-driver files to be on the your system disk (Workbench2.0). Because Workbench2.0 was full, Commodore put them on the Extras disk instead.

If you have two floppy drives, the situation is not so bad. Before you run the Font editor, for example, you should put the Extras2.0 disk into your second disk drive. Next, select the Execute Command item from the Workbench menu and enter the following:

```
ASSIGN FONTS: Extras2.0:FONTS
```

Now press the Return key or select the OK button. This command changes the location where Workbench looks for fonts from the Fonts drawer on the system disk to the Fonts drawer on Extras2.0. Now when you open the Font editor, you will see a complete list of the Amiga fonts.

With a one-drive system, your job is harder. Making a simple assignment of Fonts: to Extras2.0:Fonts doesn't work, because you still need Workbench2.0 in your drive to open the Font editor, and, contrary to the way most Amiga programs work, the editor ignores the assignment if Extras2.0 isn't in a drive when the editor looks for fonts. (Most Amiga programs would have the system put up a requester asking you to insert the Extras2.0 disk.)

The simplest solution is to assign Font: to Extras2.0:Font using the Execute Command item, copy the Font editor to the Ram Disk, and replace the Workbench2.0 disk with the Extras2.0 disk. Before opening the Font editor from

the Ram Disk, however, you need to access Execute Command again and enter the following:

```
ASSIGN ENVARC: RAM:ENV/SYS
```

Again, press Return or select OK to execute the command. This assignment ensures that you won't have to remove the Extras disk and insert the Workbench disk when you open the Font editor from the Ram Disk. With the proper assignments made and with the Extras2.0 disk in your drive, you can now open the copy of the Fonts editor on the Ram Disk and make your font selections.

For printer drivers, the solution is similar. In this case, however, the assignment statement tells the operating system where to find the printer drivers, not the fonts. The printer drivers are in the Devs/Printers drawer on Extras2.0. To direct the Printer editor there, enter the following in the Execute Command window before you use the Printer editor:

```
ASSIGN DEVS: Extras2.0:DEVS
```

One-drive owners will still have to make the second assignment for Env-Archive.

When your done with either the Font or the Printer editor, you should reset the assignments to the defaults. Using the Execute Command item, you reset the Fonts: assignment by entering:

```
ASSIGN FONTS: SYS:FONTS
```

For the printer drivers, you enter:

```
ASSIGN DEVS: SYS:DEVS
```

Permanent Solution

The problem with these fixes for one and two drive systems is that they don't let you make permanent changes to your font and printer selections. Sure, you can select the Save button in the editors, but the changes will not survive a reboot of your system. Even on a two-drive system with Extras2.0 is in the external drive, your system won't find the fonts or printer drivers when you reboot unless you modify your Startup-sequence, which you don't know how to do (yet). On one-drive systems, the Save button merely saves your selections to the Ram Disk, which is wiped out with a reboot.

The best solution to both problems is to make room on your Workbench2.0 disk for all your fonts and the driver for your printer. So what do you move off

of your Workbench2.0 disk to free space? You have lots of possibilities, from the narrator.device to little-used AmigaDOS commands, but I recommend that you move the Preferences editors from Workbench2.0 to Extras2.0. That may seem like a radical solution, but it really isn't. The editors do look in specific places for their current settings and presets, but the editors themselves don't have to be in a specific drawer or disk. They are natural candidates for migration to another disk.

With a two-drive system, moving the editors is easy. Put Workbench2.0 in one drive and Extras2.0 in another. Open the Workbench2.0 icon and drag the Prefs drawer over the Extras2.0 icon. When you release the button, Workbench will copy the Prefs drawer and its contents to Extras2.0.

Copying from one disk to another on a one-drive system is a little harder and usually requires a lot of disk swapping. To avoid that, open the Workbench2.0 disk and drag the Prefs drawer to the Ram Disk icon. Next, replace Workbench2.0 with Extras2.0, open the Ram Disk, and drag the Prefs drawer from the Ram Disk to the Extras2.0 icon.

With a copy of the Prefs drawer on the Extras2.0 disk, it's time to clean up. With a one-drive system, you should first go into the Ram Disk, select the Prefs drawer, and delete it using the Icon menu's Delete item. With both one-and-two drive systems, you should open the Prefs drawer on the Workbench2.0 disk (put it back in a drive if you removed it). Select one of the editor icons, and then shift select all the others. Do not select the Presets drawer or the Env-Archive drawer, if it is visible. Go to the Icons menu and select Delete. When the confirmation requester appears, make sure that you're deleting 13 files and no drawers before selecting OK. Whew! You now have room on your Workbench2.0 disk for the fonts and one printer driver.

Copying them over is fairly easy. With a two-drive system, put Workbench2.0 in one drive and Extras2.0 in the other, open both disk icons, and choose Show All Files from the Window menu for each of the disk windows. Now, drag the Fonts drawer from the Extras2.0 window to the Workbench2.0 window. This will replace Workbench2.0's Fonts drawer with a copy of the one on Extras2.0. Be careful that when you drag Fonts drawer icon to the Workbench window that you don't release the button while the icon is over any other drawer. You want to copy to the main Workbench2.0 window, not to one of its drawers.

Copying the printer driver you selected with the Printer editor is only a little more involved. First, with Show All Files still active on both disks, open the Devs drawer on each disk, and the Printers drawer inside each Devs window. Next, drag the icon of your printer's driver from the Printer window of

Extras2.0's Devs window to the Printer window of Workbench2.0 Devs window.

With a one-drive system, you're going to do the same things: Drag Extras2.0:Fonts to Workbench2.0 and drag your driver from Extras2.0:Devs/Printers to Workbench2.0:Devs/Printers. To avoid disk swapping, you should use the Ram Disk as a go-between by first dragging the appropriate drawer and driver from Extras2.0 to the Ram Disk, swapping Extras2.0 for Workbench2.0, and then dragging the drawers from the Ram Disk to their destinations on the Workbench2.0 disk.

Commodore has been known to change the number of files on its release disks, and so you may find that not all your fonts will fit on Workbench2.0 or that you later need room for other things. In this case, you should delete some of the fonts you don't want or like to make more room. Later, as you learn more about your system, you will discover other ways to make room on Workbench2.0.

Deleting a font is tricky. The actual fonts are the files with names such as 13 and 18 that you find in the drawers named Helvetica, Times, and so on, within the Fonts drawer. The files named Times.font and garnet.font simply describe the sizes of the fonts contained in the numbered files in the Times and garnet drawers, respectively. Deleting a .font file means that you can't access the group of fonts to which that file points. To get rid of individual fonts, you have to go into the font drawers and delete the files you don't want. Note that if you delete only a few of a drawer's fonts (for example, deleting 11 and 13 from the Courier drawer while keeping 15, 18, and 24), you'll have to run the FixFonts tool described in the next chapter.

Conclusion

The power the Amiga Preferences system gives you to customize your computer working environment is unprecedented in personal computers. Don't be afraid to experiment and make mistakes. There is no damage you can do with the Preferences editors that can't be undone. After all, you still have your original copies of Workbench2.0 and Extras2.0 tucked away in a safe place, don't you?

– 5 –

The Contents of Workbench

Your system disk contains a lot more than the Prefs drawer. This chapter examines the Workbench tools available to you on your Workbench2.0 disk, while Chapter 6 describes the Workbench tools available on the Extras2.0 disk. If you have a hard-drive system, the contents of these two disks are combined in your hard drive's System2.0 partition. You should have no trouble, however, following the discussion.

Figure 5-1 shows the windows of the Workbench2.0, Extras2.0, and System2.0 disks. Disregarding Trashcans, the Workbench2.0 and Extras2.0 disks together contain seven drawers and one project. The System2.0 disk has eight drawers and one project. Hard-drive systems use the extra drawer — Monitors — to add monitors to the system via an AmigaDOS command in the Startup-sequence. This lets you add monitors without starting Workbench. Because Workbench lets you add monitors by moving their icons to the WBStartup drawer, the absence of the Monitors drawer on the floppy-disk release of Workbench makes no difference in performance.

Apart from this minor difference, both floppy- and hard-disk-based systems come with the same set of drawers which, to a large degree, have the same contents. As we examine the drawers, I'll point out any differences between floppy- and hard-disk systems.

First, note one project — Shell — is available at the root level of the System2.0 and Workbench2.0 disks. This project and the functions of its default tool, CLI, are the subject of Chapters 7 through 10, so for now I'll skip over the Shell and the related iconless drawers.

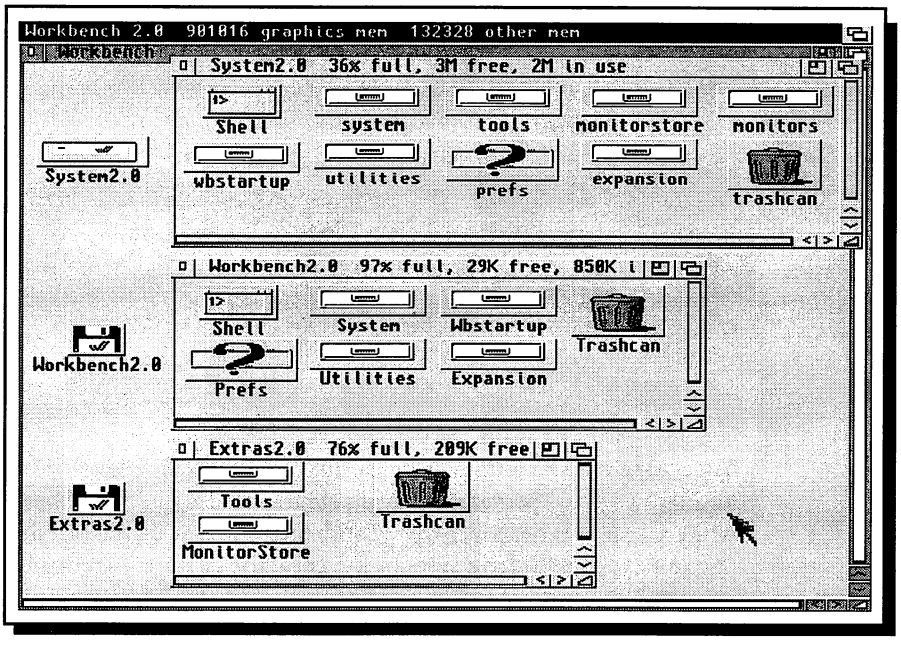


Figure 5-1 Amiga OS 2 Volumes

If you have a hard-disk system, the System2.0 volume holds the contents of the Amiga OS 2 release. Due to the 880K limitation of floppy disks, the floppy version of Amiga OS 2 takes two volumes, Workbench2.0 and Extras2.0.

The System Drawer

As you might expect, the System drawer contains tools that affect the operation of your system. The nine tools are: CLI, AddMonitor, BindMonitor, SetMap, Format, DiskCopy, NoFastMem, RexxMast, and FixFonts. Many of these are vital to the operation of your computer. For example, the Workbench interface calls the Format tool when you choose the Format Disk option from the Icons menu. If you drag the icon for one disk over that of another, Workbench calls the DiskCopy tool.

If you rename the System drawer or move it out of the system disk's root window, Workbench cannot access these tools and make their functions available to you. All in all, it's best not to mess with the name and location of this drawer or its contents. Now, let's look at the contents in alphabetical order.

AddMonitor

This tool lets you inform the system what type of monitor you are using with your Amiga. The default monitors are NTSC monitors — such as the Commodore 1084S — in North America and PAL monitors elsewhere. Support for your default monitor is built into the system.

To add a new monitor, open the MonitorStore drawer. (MonitorStore is on Extras2.0 on floppy-drive systems.) Each of the monitor icons in MonitorStore is actually an AddMonitor project. Double-clicking on one of these adds the monitor to the system list.

To automatically add a monitor at startup, move a monitor icon from MonitorStore to the WBStartup drawer. WBStartup has a special property: Workbench opens anything in it when you start the machine. Because the monitor icons list AddMonitor as their default tool, opening them causes the system to run AddMonitor. AddMonitor doesn't open a window or create its own menus. It does its job and exits without any fuss. You see its results when you open the Overscan and ScreenMode editors and find the new modes available.

BindMonitor

While AddMonitor serves a vital system function, BindMonitor provides the finishing touches by assigning names to the Amiga's display modes. Internally, your system uses hexadecimal numbers to keep track of the different modes. BindMonitor equates these numbers with understandable names.

BindMonitor uses the Mode_Names project found in the WBStartup drawer. If you select this project and then choose Information from the Icons menu, you'll see a list of the Tool Types that Mode_Names and, by extension, BindMonitor can recognize (see Figure 5-2). With the exception of modes (such as Lores) that aren't available for the Workbench screen, the names in the Tool Types list correspond to the names used in the ScreenMode editor. To see how BindMonitor works, select the Tool Type for NTSC:Hires from the list. When it appears in the text gadget below the list, edit the name to NTSC:George and press the Return key. Exit the Information display by selecting Save. Now, invoke BindMonitor by double-clicking on the Mode_Names project, and open the ScreenMode editor. Notice that you now have a display mode called NTSC:George. If you think Hires is a more descriptive name for this mode, you can repeat the process and edit George back to Hires.

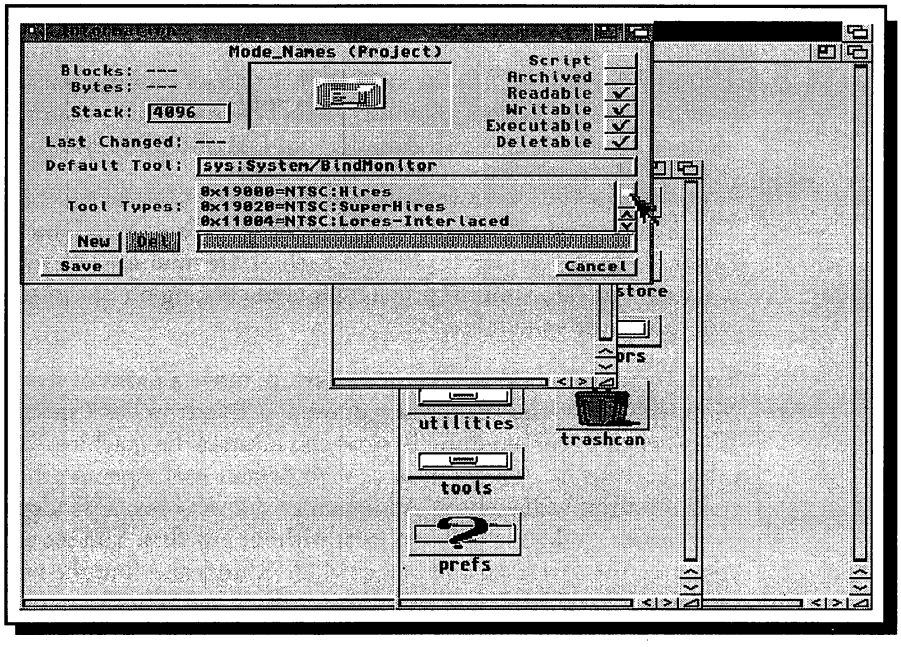


Figure 5-2 Mode_Names Tool Types

The information display of the Mode_Names project reveals how BindMonitor assigns English-like names to the internal representations of the Amiga's graphics modes.

Because BindMonitor recognizes all the Tool Types listed in the Mode_Names project, you can change any name by adding the appropriate Tool Type into BindMonitor's own Tool Type list. For example, select BindMonitor and choose Information from the Icons menu. Enter the following into the BindMonitor Tool Type text gadget:

0x19000=NTSC:George2

Press Return and select the Save button. Next, double-click on BindMonitor. Finally, open the ScreenMode editor to see your new George2 display mode.

If you don't pass any Tool Type to BindMonitor, either through a project or its own Information window, it has no affect on the mode names currently in use. If the Mode_Names project is missing from the WBStartup drawer when you boot your system, you get the visible size numbers of the modes instead of mode names in the ScreenMode editor list.

BindMonitor is not a very important tool, and you'll probably never access it directly. It demonstrates, however, how far Commodore has gone to attend to every detail in the design of Amiga OS 2.0.

The CLI and RexxMast Tools

The CLI (command line interpreter) is the default tool of the Shell project. The subject of the second part of this book, it accepts typed commands and calls the appropriate programs to execute the command.

Through Amiga OS version 1.2, CLI was the name of the Amiga command-oriented interface. In version 1.3 the CLI was joined by the Shell, a more powerful version of the original CLI program. With Amiga OS 2.0, the CLI and Shell have become synonymous. Because the 2.0 command interface is an outgrowth of the 1.3 Shell program, I will refer to the AmigaDOS 2.0 command interface as the Shell.

Double-clicking on the RexxMast (short for Rexx Master) tool loads the ARexx interpreter into memory, where it is available to you and to applications programs that use ARexx. If you've already loaded ARexx, opening this tool will produce a message that ARexx is already active on your system. Normally, the Startup-sequence file starts ARexx when you boot Workbench2.0 or System2.0.

DiskCopy and Format

DiskCopy is the tool that Workbench uses when it copies a disk. If you drag one disk icon onto another or select a disk icon and choose Copy from the Icons menu, Workbench calls this program. You can also copy a disk by selecting a disk icon, and then double-clicking DiskCopy while holding down a Shift key. If you open the DiskCopy icon without selecting a disk icon, the system tells you to access the DiskCopy item in Icons to copy a disk.

DiskCopy isn't designed for use with hard disks. Just for fun, I selected the icon for a 30 MB partition on my hard drive and, with a Shift key depressed, double-clicked on DiskCopy. After about a minute, I received a message that the copy would require over 97 million disk swaps. I decided to cancel the copy operation.

Before you can use a disk on your Amiga, you have to set up the electronic signposts that AmigaDOS uses to find and store files on the disk. Called formatting or initialization, this process is the job of the Format tool.

Format is the same tool that Workbench accesses when you select the Format Disk item from the Icons menu (see Chapter 3). You open the Format tool by selecting a disk icon and shift double-clicking on the Format icon. Note that you don't have to format a new disk before copying an entire disk to it using

DiskCopy. In addition to copying the files and drawers, DiskCopy reproduces the formatting of the source disk.

For the most part, you'll access DiskCopy and Format through the Icons menu rather than directly. Do not, therefore, move or rename these files.

The FixFonts Tool

By definition, a font is a collection of characters that are the same size and share the same typeface. To speak of the Times font is incorrect; Times is a typeface. Times 24 is a font. Your Amiga stores fonts with two files. One file has a .font ending and describes the sizes of the font available to you. A second file, contained within a directory named for the typeface, holds the actual description of the font. This second file is always named by a number, such as 24 or 11. The combination of the .font file and the number file describes a font.

Often, you may want to delete little-used fonts from or add a font size or a new typeface to your system. Whenever you do this, you have to run the FixFonts tool to ensure that the entries in the .fonts files match the fonts available in the font drawers. FixFonts doesn't open a window or create any menus. It simply makes sure that the information contained in the .font files matches the actual fonts in the font drawers.

NoFastMem

You can have two kinds of memory in your system: chip RAM (also called graphics memory), which stores the information that creates the output display, and fast RAM, which stores everything else. Chip RAM is so called because it is accessible by both the central processor and the Amiga custom chips. Your system must always contain some chip RAM, but fast RAM is optional.

The first Amiga model — the A1000 — came with 256K bytes of memory, expandable to 512K, all of which was chip RAM. In fact, Amiga OS 1.2 was the first version to contain the routines that let you add Autoconfig fast RAM expansion boards to your Amiga. Many early programs assumed that all the memory in your machine was chip RAM. Thus, they didn't explicitly assign things to chip RAM that they should have. Because the Amiga OS assigns data to fast RAM unless an explicit assignment is made, some information that belonged in chip RAM ended up in fast RAM instead. The most frequent oversight was pointer imagery. Most programs provide their own data for the pointer. When fast RAM expansion boards first appeared, it wasn't uncommon to load a program and be unable to run it because you couldn't see the pointer, which had been stuck up in fast RAM by the operating system.

NoFastMem lets you run these early programs by deleting all fast memory from your system's memory list. Most commercial programs fixed this bug years ago, but you might occasionally find a program you want to run that doesn't allocate chip RAM properly. In that case, double-clicking on NoFastMem before starting the program should let it run properly.

If you have an unexpanded Amiga 500, you will never have to use NoFastMem because all your memory is chip RAM. Only when you have fast RAM on your system might you encounter this problem. You can see the results of running NoFastMem in the menu bar of the Workbench screen: It changes the value for "other mem" to 0.

SetMap

Commodore is an international company and sells Amigas all over the world. In fact, there are more than twice as many Amigas outside of the United States as there are inside. The SetMap tool lets you configure your keyboard to conform with different languages. The Amiga supports 15 different mappings of the keys on its keyboard. One mapping — usa1 — is built into ROM; the other 14 are supplied on disk with Amiga OS 2.0.

On a hard-disk system, you find the alternate keymaps in the Keymaps drawer, which resides in the Devs drawer of the System2.0 disk. (You need to first activate the Show All Files option to see them.) If you're using a floppy-based version of OS 2.0, your Workbench2.0 disk also has a Keymaps drawer inside its Devs drawer, but it is empty. To save room on the disk, the keymaps are stored in the Keymaps drawer in the Devs drawer of the Extras2.0 disk. Before you can use a keymap other than usa1, you must move it from Extras2.0 to the Keymaps drawer on Workbench2.0. With Show All Files active, this is easy; just drag the appropriate icon from Extras2.0:Devs/Keymaps to Workbench2.0:Devs/Keymaps.

Figuring out the Amiga keymap files' short, cryptic names is a little tougher. Here is a list of the available files and the languages or countries they support.

cdn	French Canadian	*gb*	Great Britain
ch1	Swiss French	*i*	Italian
ch2	Swiss German	*is*	Icelandic
d	German	*n*	Norwegian
dk	Danish	*s*	Swedish
e	Spanish	*usa0*	United States/Canada
f	French	*usa2*	Dvorak, United States/Canada

The usa0 keymap is the default included with the earliest version of Amiga OS. Commodore makes it available for compatibility purposes.

Although most Indo-European languages use the same alphabet, the different keymaps have special characters and accents, plus the proper currency character. As you will see, most characters and accents are available via the the Alt, Shift, and Control keys. SetMap simply makes the main keys reflect the language of the user.

Unlike tools such as DiskCopy and Format that you can access through a variety of ways, SetMap gives Workbench users only one way to set the keymap. First, you must move the desired keymap into the Devs/Keymaps drawer of your system disk. Select SetMap and access Information from the Icons menu. Next, click on the New button beside the Tool Type list and enter the following into the text gadget:

keymap=*yourchoice*

where *yourchoice* is the name of the keymap that you want to use (see Figure 5-3). Press Return and select the Save button to exit the Information window. To make your selection active, double-click on the SetMap icon.

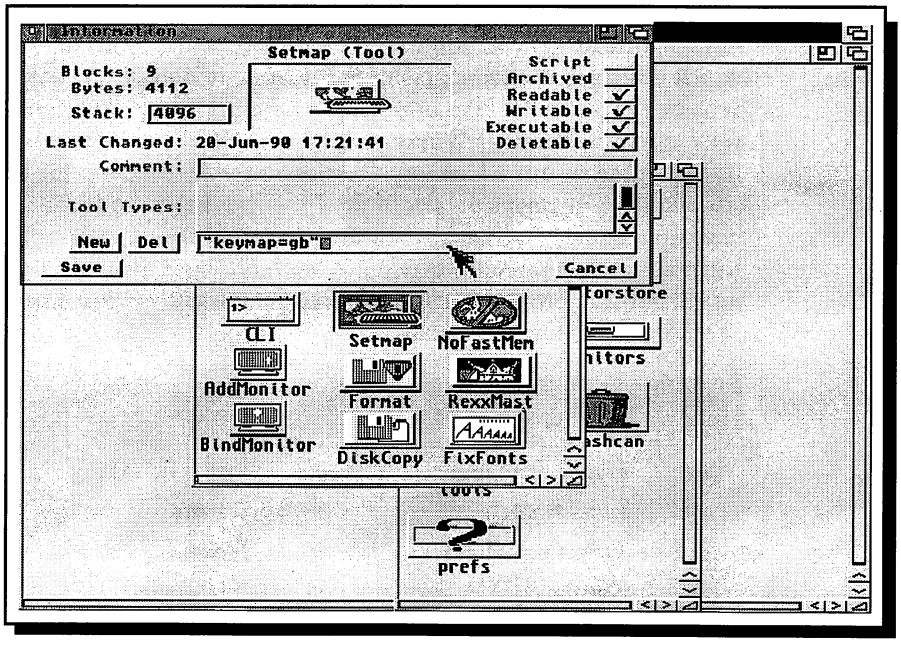


Figure 5-3 Setting a Keymap

You set a keymap for your keyboard with the SetMap tool. In the figure, the keymap is being set to gb, which stands for Great Britain.

If you want to make your keymap selection permanent, you will either have to modify your startup-sequence to run SetMap when you boot or move the SetMap tool (with the proper Tool Type set) to the WBStartup drawer. If you don't run SetMap sometime during the boot procedure, the keymap will be set to usa1. If, after booting with an alternative Tool Type, you want to switch to the default, simply enter the usa1 keymap in the Information window. Don't worry that usa1 doesn't appear in your Devs/Keymaps drawer; it is built into ROM.

The Utilities Drawer

The tools in the Utilities drawer perform more practical tasks than those in the System drawer. Three (Display, More, and Say) provide alternate ways for you to get output from files, while a fourth (Clock) tells you the time. These programs are fairly old hat on the Amiga.

New to Workbench 2.0 is a system called the Commodities Exchange. Rather than trading in pork belly futures, however, the Workbench Commodities Exchange provides an environment in which small utility programs can coexist on the Amiga without interfering with each other or with other applications running on the system. In addition to the exchange control program — aptly named Exchange — Workbench 2.0 contains four commodities utilities. As the distribution of Amiga OS 2.0 spreads, you'll see additional commodities available both commercially and through the public domain. Besides the Exchange program, the commodities available in the drawer are AutoPoint, Blanker, IHelp, and NoCapsLock.

The Clock

The Clock program displays the time from the Amiga's system clock. If your Amiga has a battery-backed clock, a statement in the standard Startup-sequence file loads the contents of the battery-backed-up clock into the system clock. Thus, the Clock should always display the correct time. If your system does not have a battery-backed clock, you will have to set the system time manually through the Time editor or the AmigaDOS Date command before the Clock registers the correct time.

When you open the clock, its window displays the time on a 12-hour clock face and an AM or PM indicator in the upper-right corner. The window uses all the standard window gadgets except the scroll gadgets. Because the program sizes the clock face dynamically to fill the output window, scroll gadgets are unnecessary. The program also has five menus. To see them, open the Clock

and click in its window. The Clock menus (Type, Mode, Seconds, Date, and Alarm) will now be visible when you press the menu button.

Type: Type's two items, Analog and Digital, are mutually exclusive; when you choose one, you automatically cancel the other. A check mark indicates your current selection.

Analog: The default value, Analog causes the Clock program to output the time using a standard clock face, with hour and minute hands, in its window.

Digital: Choosing this item reduces the output window to a small title bar that contains the time in digital form. The title bar doubles as a drag bar, allowing you to position the Clock in an unobtrusive place on your output display. Also available are a close gadget and a depth gadget.

Mode: The Mode menu lets you switch between a 12-hour clock and a 24-hour clock. The items are mutually exclusive and marked with a check.

12 Hour: With an analog clock, this displays the time on a clock face divided into 12 hours. An indicator in the upper-right of the window tells whether it is AM or PM. On a digital clock, the AM/PM indicator shares the title bar with the time.

24 Hour: More familiar in Europe and the military is the 24-hour clock provided by this item. On a digital display, it shows the hour as the number of hours since midnight. For example, 10 PM thus becomes 22:00. There is no AM/PM indicator. On an analog display, this item removes the AM/PM indicator but doesn't change the standard clock face. This item thus isn't too useful with an analog display.

Seconds: This menu has two mutually exclusive items.

Seconds On: On an analog display, this item displays a sweep second hand on the clock face. On a digital display, it adds the seconds to the time display in the title bar.

Seconds Off: Remove the second hand or the seconds digits from the output by selecting Seconds Off.

Date: The fourth menu also has two mutually exclusive items.

Date On: On an analog display, this item makes the current date appear at the bottom center of the window. On a digital display, the date alternates with the time every two seconds.

Date Off: The opposite of Date On, this item keeps Clock from displaying the current date.

Alarm: With the items in this menu, you can have Clock notify you at a specific time.

Set: The dots after this item indicate, as always, that the item brings up a requester. If you've selected a 12-hour clock type, the Alarm Set window comes up with an hour slider divided into 12 segments, a minute slider, and an AM/PM cycle gadget (see Figure 5-4). The 24-hour clock's requester lacks AM/PM gadget, but has an hour slider with 24 increments. You use the sliders and the gadget (if present) to indicate the time you want an alarm sounded.

Alarm On: This item and Alarm Off are mutually exclusive. If the clock reaches the time you set when Alarm On is active, the system emits a tone and flashes the screen. (You must have speakers hooked up to your Amiga to hear the sound.) The Clock program must still be running for the alarm to work. You can't save an alarm setting, but must reset it every time you open the Clock.

Alarm Off: Keeps the alarm from going off when the system time reaches that specified in the Set item.

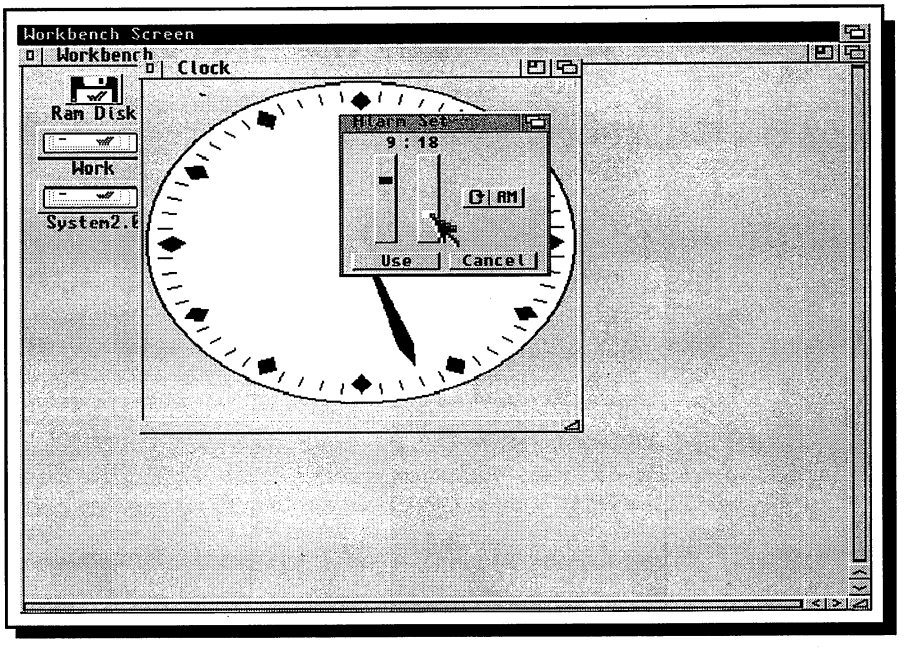


Figure 5-4 Setting the Alarm

Choosing the Set item in the Clock's Alarm menu lets you set the time when the alarm goes off. The Amiga both flashes the screen and makes a noise when the alarm goes off.

The Display Tool

The Amiga is a great computer for creating pictures, whether you use a digitizer such as Digi-View or draw them from scratch with a paint program such as DeluxePaint III. Display lets you view those picture files on your Amiga. By selecting a picture file icon, and then holding the Shift key and double-clicking on Display, the program loads and displays the picture file. If you click on the picture, the message "<- Close when title hidden, or Control-C" toggles on and off in the picture's title bar (see Figure 5-5). The message tells you that when it is not visible, you can close the picture by selecting the invisible close gadget in the upper-left corner of the screen or by pressing CTRL-C.

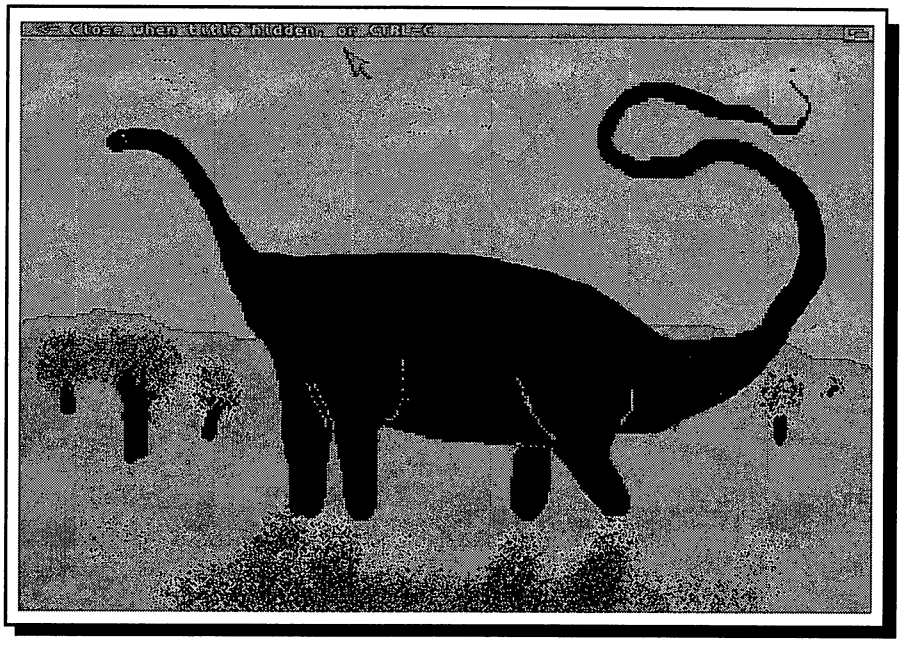


Figure 5-5 Closing a Picture

To close a picture you're displaying, you click in the upper-left of the display screen while the message is not visible. Clicking within the picture toggles the appearance of the message.

If you extend select several files, Display will cycle through them one at a time. You determine when the next picture comes up by clicking either mouse button or by closing the current picture with CTRL-C. The pictures continue cycling until you enter CTRL-D. You can print a picture at any time by pressing CTRL-P.

You can control the behavior of Display by altering its Tool Types through the Icon menu's Information item. The first four Tool Types can appear either in Display's Tool Type list or in the list of a FileList project. The fifth Tool Type is for a FileList project only, and the last three can appear in the Tool Type list of any picture file you want to view.

Timer=<n>: The first Tool Type lets you determine how long you want to display a picture before the program closes it. Here, <n> equals the display time in seconds.

Mouse=: A Boolean Tool Type, Mouse= can be set to either True or False. When True, it lets you use close a picture by clicking either mouse button. Note, however, that it only works when you've selected multiple files for viewing. As with any Boolean Tool Type, the absence of this item from the Tool Type list is the equivalent of setting the item to False.

Loop=: When you've set this Boolean Tool Type to True and selected multiple picture-file icons, Display will continuously cycle through the pictures until you press CTRL-D. When set to False or when absent, Display shows each picture only once before exiting.

Print=: When set to True, this Boolean Tool Type tells Display to dump the currently displayed picture to your printer.

Back=: You use the Back=True setting to display your pictures behind the Workbench screen. This may seem odd until you realize that it takes much less time and memory to print a picture that isn't in view than it does to print one that is. You should use Back in conjunction with Print.

FileList=: When set to True, this Tool Type identifies a FileList project. A FileList is simply a file containing a list of the names of the picture files you want to view with Display. To create such a project, you first use a text editor or word processor to create a file of names, then save it to the drawer that contains the picture files. If your word processor or editor creates an icon for this file, you select the icon, call up the Information item, select the New button on the Tool Type list gadget, and enter:

```
FileList=True
```

You must then change the default tool to:

```
SYS:UTILITIES/Display
```

If, like AmigaDOS' Ed editor, your text editor does not create an icon file when you save a file, you must use the IconEditor tool in the Tools drawer to

create a new project icon and save it with the same name as your file list, adding a .info extension. See the section on the IconEditor for more information.

EHB=: Used in the Tool Type list of a picture file icon, this Boolean item tells Display that the picture should be displayed in Extra_Halfbrite mode, which doubles the number of colors in the picture.

NOTRANSB=: When set to True, this Tool Type informs Display that the picture shouldn't be displayed with a transparent border region.

Video=: When set to True, the final Tool Type informs Display that it should use overscan and a transparent border to display the picture file.

With its ability to read a list of files contained in a FileList project, Display lets you create simple slide shows that you can automate with the Timer Tool Type. You should be careful, however, when using the Back option. If you use the Workbench screen depth gadget or the command keys to bring a Display screen to the front, you may be unable to return to Workbench.

The More Tool

Like Display, More is a file viewer. The difference is that More lets you view text files. You can choose a file to view in two ways. If the file has an icon, you can select the icon and then, while holding down the Shift key, double-click on More. The second method is to open More directly.

Double-clicking on More without a file selected brings up the system file requester. Here, you can choose any file on any disk in your system for viewing. The details of operation of the system file requester are discussed in the section on the default Preferences menus in Chapter 4.

More is intended for viewing ASCII text files. ASCII text uses seven bits of an eight-bit byte to store a character. Most text editors and word processors produce ASCII text files that can be viewed using More. Many other Amiga files such as program files and picture files are not text files but binary files. When More detects that a file may contain binary data, it warns you and gives you the option to quit. At this point, you press q to quit or the spacebar to continue. If you continue, More tries to interpret the binary data as best it can. Most often, this means printing some characters from the Amiga's alternate character set. This output doesn't often make sense (see Figure 5-6).

Once you've loaded a text file into More, the program displays the contents 22 lines at a time. If you're using an interlaced screen, this fills just half your screen. You can use the drag sizing gadget on the More window to open it to fill your display.

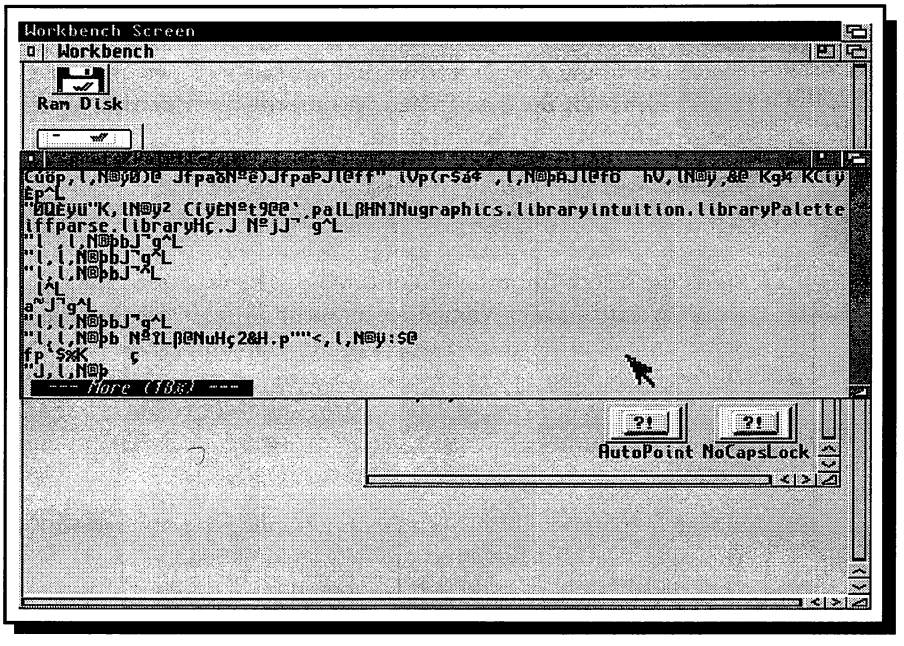


Figure 5-6 Viewing Binary Files with More

The More utility is designed to work with text files only. When you load a binary file such as a program, it tries to display the text equivalents of the binary numbers. Most of these equivalents are from the alternate character set.

At the bottom of the More window is a message indicating the percentage of the file you've viewed so far. You move about in the file by using the following More commands.

Spacebar: Pressing the spacebar while viewing a file advances you one page in the file. A page is the number of lines of text that More can display at once. You can use the sizing gadget to alter the page size. Pressing the spacebar with the message "End-of File" showing will close the More window.

Return: Pressing the Return key advances you one line in the text file. If you're at the end of the file, pressing Return closes the window.

Backspace: Hitting Backspace moves you up one page in the file, letting you scroll backwards.

q: Pressing q at any time closes the More window. It has the same effect as selecting the close gadget or pressing CTRL-C.

<: Pressing the less-than key returns you to the beginning of the file.

>: The greater-than key moves you to the last page of the file.

%<n>: Entering the percent key followed by <n>, where <n> is a number from 0 to 100, scrolls the display that far into the file. Entering a value outside the range moves you to the end of the file.

/<search-string>: More can search a text file for a word, phrase, or group of letters. To search for a sequence of characters, press the slash key and type the <search string>, the desired group of characters. When you press Return, More starts the search from the top of the file. The slash preceding the string initiates a case-sensitive search, which requires the string in the text file to have the same capitalization as well as spelling to match the search string. Thus, /George would not produce a match with george or GEORGE. Upon finding the first occurrence of the string, More scrolls line containing the string to the top of the window. If it can't match the string, it displays the message "Not Found."

.<search-string>: Starting with a period initiates a noncase-sensitive search. In this case, .workbench would produce a match with all of the following: WORKBENCH, workbench, wOrkbench, Workbench, and so on.

n: After you've started either type of search, press the n key to scroll the next occurrence of the search string to the top of the display.

h: This most useful of the More command keys brings up a list of the utility's commands. Pressing the Help key has the same effect. To return to the More window, press any other key.

CTRL-L: If the More window has not refreshed properly after an overlapping window or requester has been removed, you can force a refresh by pressing CTRL-L.

To see More in action, try looking at a file I've talked a lot about — the startup-sequence file. First, open the More icon. When the file requester pops up, select the Disks button. Near the bottom of the list window, you will see a drawer named S that has an <ASN> label. Open the drawer by clicking on the S. From the list of files in S, click on Startup-sequence and then select the OK button. More now loads the Startup-sequence file and displays it. You should see about 70 percent of the file in the first page. Use the command keys to scroll through the file and perform some searches. You can't hurt the Startup-sequence file; More is simply a file viewer, it isn't an editor.

The Say Tool

Like the Clock, Say has been around since the first release of Amiga OS. Say translates words and characters you type at the keyboard into speech. Obvi-

ously, you must have your Amiga's sound output jacks hooked up to an amplified speaker.

When you open Say, it creates a Phoneme window and an Input window (see Figure 5-7). Input, where you type what you want output as speech, must be active in order to receive input. The Phoneme window displays the *phonemes*, or parts of speech that the program translates your input into and also lets you modify the spoken output. When first opened, the Phoneme window lists the commands you can enter into the Input window to change the characteristics of the spoken output. All commands must be preceded by a minus sign (-).

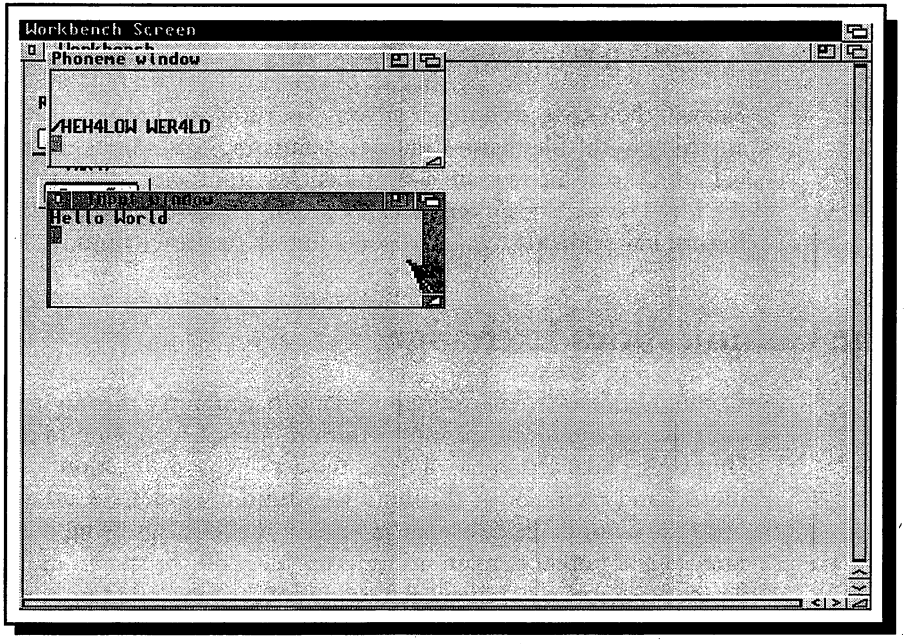


Figure 5-7 The Say Windows

Say opens two windows, one for your input and another to display the phonemes created by your input. You exit the program by entering a carriage return as the first character on an input line.

-m: This command outputs the speech using a male voice.

-f: Output uses a female voice.

-r: Output is robot-like.

-n: Output is more natural sounding.

-p<n>: Sets the pitch of the output, where <n> is a number from 65 to 320. The higher the number, the higher the pitch.

-s<n>: Sets the speed of the output, where <n> is a number from 40 to 400. The higher the number, the faster the output.

Finally, Say also lets your Amiga speak a text file. To demonstrate, let's have Say read the Startup-sequence file.

Because the file-output option is only available through a command-line interface, you must first access the Execute Command item from the Workbench menu. Now, enter the following into the text gadget:

```
SYS:UTILITIES/SAY -x S:Startup-sequence
```

Say will then say the contents of your Startup-sequence file. If it encounters anything in the file that it thinks are commands but that it can't understand, it will print error messages in a Workbench output window as it goes through the file. To exit Say, either select the Input window's close gadget or enter the Return key as the first character on a line in the Input window.

The Commodities Exchange

Commodities are small utility programs that monitor the input stream from your mouse and keyboard into Intuition. They can respond to predetermined "hot keys," take action based upon the actions — or inactions — of the mouse, and even modify the input to Intuition windows. Commodore developed the Commodities Exchange in response to the growing number of commercial and public-domain utilities that hang off the system input stream. These programs never knew about one another and often interfered with each other's operations. The Exchange provides a way for programmers to write input stream utilities that don't interfere with one another.

You start a commodity as you would any other program, by opening its icon. Commodities come in two flavors — those that open their own windows and those that don't. Commodities that open their own windows have a special capability not available to other Workbench tools; they can make their windows disappear and still remain active. You can recall these windows at any time by pressing the proper hot key combination. Note that a hot key doesn't start up a commodity, it just recalls the window of one that is already active.

For example, the Blanker commodity is a utility that blanks your display screen after there has been no mouse or keyboard activity for a certain length of time. It is an example of a screen saver; a utility that keeps your monitor's

phosphors from burning through, as can happen if the monitor display doesn't change every once in a while. When you open Blanker, it opens a window (see Figure 5-8) that contains a simple requester. This requester contains a text gadget that lets you enter the number of seconds of inactivity that Blanker must detect before shutting off the display. The default is 60 seconds. The requester also contains two buttons, Hide and Quit. Quit closes down the Blanker program. Hide, on the other hand, closes the window but keeps Blanker active. You won't find its window anywhere on the Workbench, but if you don't hit a key or move the mouse for 60 seconds, Blanker will shut down your display.

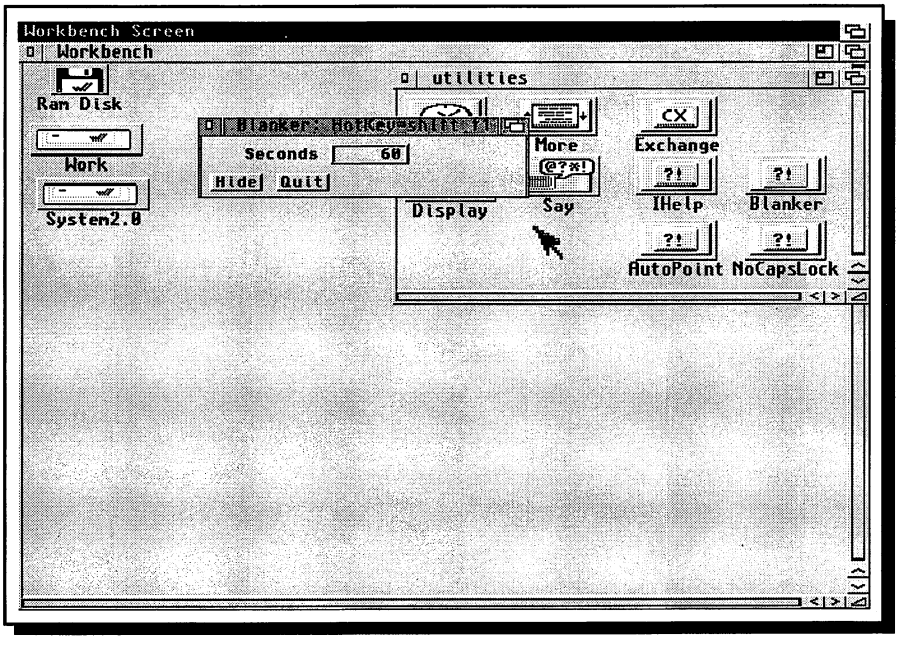


Figure 5-8 The Blanker Commodity

Like all commodities that produce display windows, blanker displays its hot-key combination in its title bar. You can change this by using the CX_POPKEY Tool Type.

The Blanker window also displays another common commodity feature: a hot key. In the title bar of the window is the message "Hot Key=shift f1." This means that if you want to recall the Blanker window while it is hidden, you need only press one of the Shift keys and the F1 key together. Note that a hot key will only bring a commodity to the front of the Workbench screen. If you're working on another screen, you'll have to return to the Workbench screen to see the window.

If a commodity doesn't open a window, you can always shut it down by double-clicking on its icon again — commodity icons act like toggles — or through the Exchange commodity.

Commodity Tool Types

In addition to any private Tool Type that a commodity supports, all commodities support the CX_PRIORITY Tool Type. In addition, commodities that open their own windows also support the CX_POPKEY and CX_POPUP Tool Types. (The commodities programs also support the WBStartup Tool Types. See the discussion of the WBStartup drawer below for a discussion of these.)

CX_POPKEY: This Tool Type lets you define the hot key for a commodity that opens a window. All commodities come with a predefined hot key. As the number of commodities available grows, you may find yourself with one or more that use the same hot-key combination. CX_POPKEY lets you change a commodity's hot key. A hot key cannot simply be one of the characters from your keyboard. If must be either a function key, or a character key pressed in conjunction with one more of the qualifiers listed below:

Alt	either Alt key
RAlt	right Alt key
LAlt	left Alt key
Shift	either Shift key
RShift	right Shift key
LShift	left Shift key
RCommand	right Amiga key
LCommand	left Amiga key
Control	Control key
Numericpad	a key on the numeric keypad
Rbutton	the right-mouse button
Lbutton	the left mouse button

For example, if you wanted to change the hot key for Blanker so that its window came up when you pressed the Right Alt, Control, and B keys together, you'd enter the following into the Tool Types gadget in Blanker's Information window:

```
CX_POPKEY=RAlt Control B
```

Selecting the Save button preserves your choice. Now, the next time you run Blanker, you'll be able to pop its window with the hot key you've defined.

CX_POPUP: This is a Boolean Tool Type. When set to True, CX_POPUP causes a commodity that uses a window to open its window when you start the commodity. Otherwise, the commodity begins with a hidden window. The only time you'd probably want to set the Tool Type to False is when you place a commodity in the WBStartup window. You may not want a bunch of windows jumping out at you when you boot your machine.

CX_PRIORITY: All Commodities Exchange programs are constantly monitoring the stream of signals between your input devices and Intuition. Some programs may be looking for the same input events or respond to the same hot key combination. To avoid potential conflicts, you can assign priorities to the commodities. Then, if two commodities are waiting for the same input event, only the one with the higher priority will see the event and respond to it. For example, if you assign Blanker and Exchange the same hot key and give Exchange a higher priority than Blanker, the hot key will always bring up the Exchange window.

You can set the priority of a commodity to any value from -128 to 127. Priorities shouldn't be a big concern unless you begin to amass a significant number of commodities.

The Exchange Commodity

The most significant commodity supplied with Amiga OS 2.0 is the Exchange program. It serves as controller for itself and the other commodities active on your system.

When you open Exchange, or call it with its hot key, it presents the window shown in Figure 5-9. In addition to the name Exchange, the title bar lists the program's currently active hot key. You can change the hot key using the CX_POPKEY Tool Type. In the center of the window is a scrolling list of commodities that are active (available) on the system. Note that Exchange is listed; you can control it as you would any other commodity. You select a commodity to work on by clicking on its name in the list. Below the list is a three-line display that carries information about the currently selected commodity. Title tells you the selected commodity's name, while Description gives you an idea of the commodity's function. Status indicates whether the commodity is enabled or disabled.

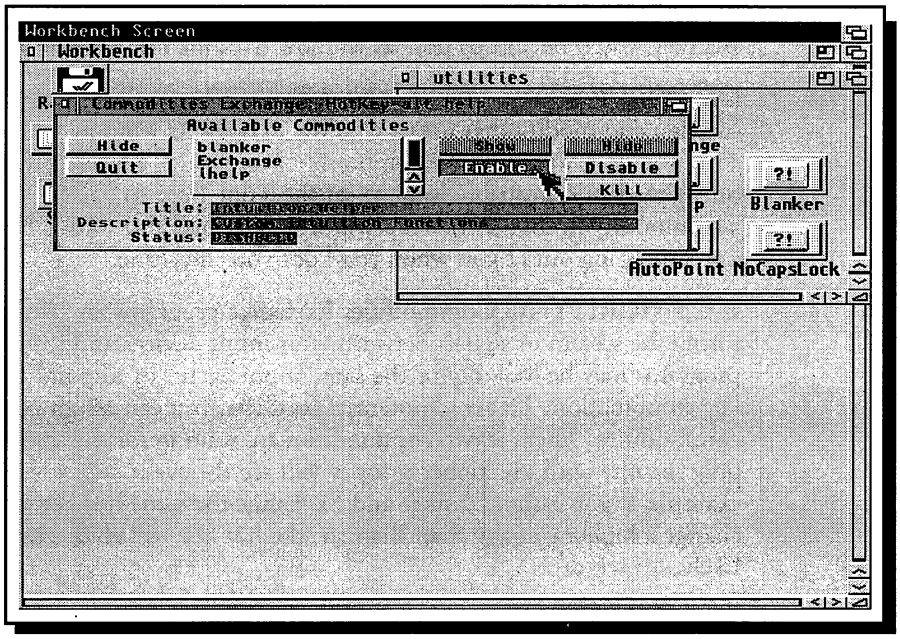


Figure 5-9 The Exchange Commodity

From the window of the Exchange commodity, you can activate and deactivate any commodity currently running on the system, including the Exchange commodity.

To the right of the list window are five buttons that let you control the currently selected commodity. If the selected commodity opens a window, all five buttons will be available, otherwise, the Hide and Show buttons will be ghosted.

Show: The equivalent of hitting the commodity's hot key, the Show button pops up the window of a commodity. Once the window is open, you can modify the commodity's operation. For example, if you called up the window for Blanker, you could change the length of system inactivity required before Blanker shuts down the display.

Hide: Selecting this button hides the window of the currently selected commodity, even if the window wasn't opened with the Show button.

Enable: This button reactivates a previously disabled commodity.

Disable: If you want to temporarily suspend the activity of a commodity without terminating the program, select Disable. This is useful when you want to run a program that might be interfered with by the action of the commodity.

A disabled commodity doesn't respond to its hot key. However, you can always bring up the window of any commodity — even a disabled one — by double-clicking on its icon.

Kill: This button shuts down the activity of the selected commodity and removes it from memory.

The Hide and Quit buttons to the left of the list window control the Exchange itself. Hide, of course, hides the Exchange window, and Quit closes down the Exchange program.

You don't have to have Exchange active to use any of the commodities. Note, however, that because you can always access the other commodities through Exchange, you don't have to remember the hot keys of any other commodity.

AutoPoint

With the AutoPoint commodity active, and neither mouse button depressed, the window under the pointer is always selected as the active window. When you move the pointer from over one window to another, the new window will automatically be selected. You start up AutoPoint by double-clicking on its icon. Because it doesn't open a window, you must either double-click on its icon again or use the Exchange program to kill it.

Blanker

I discussed Blanker at length in the Commodities Exchange section above, so I won't repeat myself here. Suffice it to say, Blanker performs its magic by disabling the Amiga output circuitry. To see your display again after Blanker has removed it, simply hit any key or move the mouse. Blanker's default hot key is Shift-F1. Because monitors are expensive devices, I recommend you use this commodity.

IHelp

For typing-intensive applications, having to constantly take your hands off of the keyboard to use the mouse can be both slow and bothersome. The IHelp commodity lets you perform five common Intuition operations with the functions keys instead of the mouse (cycling between nondrawer windows, enlarging a window, shrinking a window, cycling between screens, and zooming a window). These functions are currently set to the first five function keys, but

you can change any of the key assignments used by using the IHelp Tool Type display. IHelp does not open a window.

CYCLE=f1: With this Tool Type activated, you can select successive Workbench applications windows by pressing the F1 key. This function cycles between all open applications windows on the topmost screen. Note that cycle doesn't work with disk and drawer windows, only with windows opened by such programs as the Clock and the Shell.

MAKEBIG=f2: Pressing the F2 key with the MAKEBIG option active makes the currently selected window grow as large as it can without moving its upper-left corner. The effect is the same as dragging the sizing gadget as far right and down as the screen will allow.

MAKESMALL=f3: This Tool Type instructs IHelp to make the currently selected window as small as possible when you press the F3 key. It performs the same function as dragging the sizing gadget as far as you can up and to the left.

CYCLESCREEN=f4: I find this to be the most useful IHelp function. By hitting F4, you can cycle through all the screens on your system. This is the fastest and easiest way to move between different screens. The effect is the same as hitting successive screen-depth gadgets.

ZIPWINDOW=f5: Pressing F5 performs the same function as selecting the zoom gadget in the currently active window.

NoCapsLock

The final commodity supplied with Amiga OS 2.0 is NoCapsLock. When active, it disables the Caps Lock key, although the Shift keys continue to function normally. NoCapsLock does not open a window; you close it either using Exchange or by double-clicking again on its icon. It performs as advertised.

The Expansion Drawer

The Expansion drawer does not contain any tools, but is very important in configuring your system. The system accesses it whenever you boot.

As I mentioned in Chapter 1, the Amiga OS can auto-configure; it polls expansion hardware to discover what you've added to your system and then assigns the hardware to a specific address space in the system memory map. This

is fine for memory expansion cards, but other types of hardware such as hard-disk drives and serial boards usually require a special software program called a device driver to work correctly. Device drivers are similar to printer drivers; they convert Amiga commands into commands the expansion hardware can understand.

If you buy a hardware expansion device that attaches to the expansion port of an Amiga 500 or slides into a slot in the Amiga 2000 or 3000, the manufacturer may supply a device driver on disk with the hardware. You should move this device driver into the Expansion drawer.

As part of the boot procedure, AmigaDOS issues a Binddrivers command that searches the Expansion drawer for device drivers it can match with the Autoconfig devices attached to the machine. When it finds a match, it binds the driver to the device, ensuring that AmigaDOS will be able to communicate with the device. Do not delete, move, or rename the Expansion drawer!

The WBStartup Drawer

Another drawer that serves an important function at boot time is the WBStartup drawer. Whenever your system initializes the Workbench interface by running the LoadWB program, it executes any projects and tools that you've put into the WBStartup drawer.

When you first install Amiga OS 2.0, WBStartup contains one file, the Mode_Names project. Mode_Names is a project of the BindMonitor program, which assigns descriptive names to the various Workbench screen modes. One of the first things you should do after you install 2.0 is to move the appropriate monitor icons from the MonitorStore drawer to the WBStartup drawer. These icons are projects of the AddMonitor program. Workbench uses them to determine which modes it should make available to you through the ScreenMode editor.

You can put your own projects and tools into the WBStartup drawer, but you've got to watch out for a common pitfall. Unless you indicate otherwise, Workbench completes the execution of each project or tool in the WBStartup drawer before proceeding with the next one. If you move a project from your favorite word processor into the drawer, intending to use it after you boot, Workbench expects a signal from the program saying that it has ended before it proceeds with the next item in the WBStartup drawer. Eventually, Workbench will proceed without hearing back from the system, but if you later close the word processor, you can crash the system. The solution is to take advan-

tage of the DONOTWAIT, STARTPRI, and WAIT Tool Types that Workbench recognizes in regard to items in the WBStartup drawer.

DONOTWAIT: When this Tool Type appears in a project or tool in the WBStartup drawer, it tells Workbench not to wait for a return message from the program before launching the next WBStartup program. In fact, DONOTWAIT alleviates the program of having to inform the WBStartup code when the program exits. DONOTWAIT doesn't take any arguments. So if you want your favorite word processor ready and waiting after your machine boots, be sure to add the DONOTWAIT to the project or tool icon that you put into WBStartup. Note that all of the commodities in the Utilities drawer already have this Tool Type set.

STARTPRI=<n>: If you want the projects and tools in the WBStartup drawer to run in a certain order, you assign them different start priorities. STARTPRI uses a number, <n>, from -128 to 127 to determine which programs to start first. The tool or project with the highest priority gets started first. If STARTPRI isn't set, the priority of the tool or project is set to 0.

WAIT=<n>: Sometimes, you may want to ensure that a project or tool is completed before you run another one. The WAIT Tool Type lets you specify how long Workbench should wait after opening the current project or tool before opening the next one. You specify the length of time, <n>, in seconds.

WBStartup is especially useful for running such utilities as the commodities managed by the Commodities Exchange. Trying to run too many applications, however, will severely lengthen the time it takes you to boot your computer.

Conclusion

That wraps up our discussion of the visible contents of the Workbench2.0 disk, which, of course, includes most of the contents of the System2.0 partition. The next chapter discusses the Workbench tools available on the Extras2.0 disk and some special Workbench utilities that come with hard disk systems and the Amiga 3000.

– 6 –

Extras and Beyond

You might think that, with a name like Extras, the contents of this disk aren't very important. True, some of the tools on this disk — such as Calculator — are not vital to the smooth operation of your Amiga. Others, however, perform functions that I wouldn't want to be without. The major difference between Extras2.0 and its predecessors in earlier versions of Amiga OS is the absence of Amiga Basic, the Amiga version of Microsoft Basic.

The Extras Drawers

The Extras2.0 disk contains two visible drawers: MonitorStore and Tools. You've encountered MonitorStore before, in the discussions of the AddMonitor tool and the WBStartup drawer. This drawer has no function other than to store the projects of the AddMonitor tool until you decide to add one to the system by dragging it to the WBStartup drawer.

By default the more important drawer on the Extras disk is the Tools drawer. Depending upon whether you get Amiga OS 2.0 with an Amiga 3000, a hard-drive equipped Amiga 2000, or as a floppy-disk upgrade kit from Commodore, the contents of the Tool drawer may vary. (Notice that I use the word "may" here; Commodore's plans for Amiga 500 and 2000 upgrade kits were not finalized as this book went to press.)

To keep confusion to a minimum, I've divided the discussion of the contents of the Tools drawer into two parts. The first discusses the nine tools — Calculator, CMD, Colors, GraphicDump, IconEdit, InitPrinter, KeyShow, MEmacs, PrintFiles — that will appear in everyone's Tool drawer. After that, I describe such tools as HDToolBox that only appear in the Tool drawers of specific systems.

The Calculator

This is one of the simplest programs in the Tools drawer. Opening the Calculator brings up a small, fixed-size window titled Calc (see Figure 6-1). The Calculator is a simple arithmetic calculator that lets you enter numbers and specify functions by using the mouse, the keyboard, or a combination of the two. If you click on a number button or press a number key with the Calc window selected, that number goes into the text gadget at the top of the window. After entering a number, select an operation and enter another number if necessary. CE clears the current entry, and CA clears the calculator's memory.

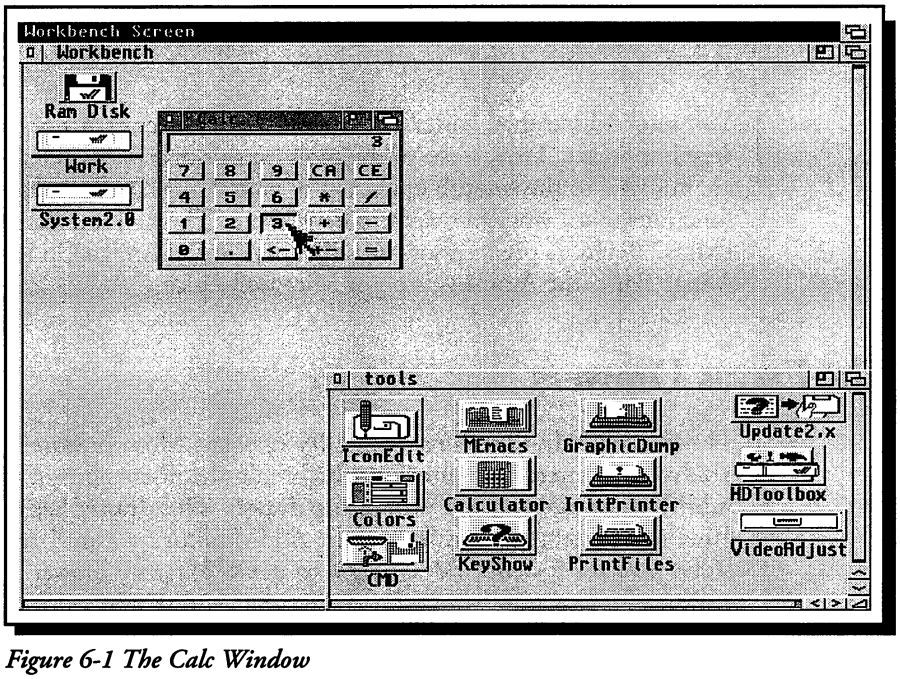


Figure 6-1 The Calc Window

Produced by the Calculator program, the Calc window consists of buttons that correspond to numbers' mathematical operators. The results of any operations you perform appear in the display window at top.

CMD

An output redirection program, CMD intercepts output normally meant for the printer and redirects it to a disk file. This can be very handy if you don't have a printer but can access an Amiga that does. In conjunction with the

GraphicDump program, it also lets you save Amiga screens to disk that you can later dump to a printer. Note, however, that you cannot load screens that were captured to disk via this method into a paint program; they are not saved in the Amiga IFF picture file format.

CMD is unusual in that it doesn't set up its own menus or open its own window. To activate it, you double-click on its icon. After it has intercepted output meant for the printer, it displays a message saying that it redirected a certain number of bytes from your parallel or serial device to disk. It then tells you that it removed itself from the system. Because it doesn't have a window or menus for input, you control the operation of CMD through its Tool Types — DEVICE, FILE, SKIP, MULTIPLE, and NOTIFY.

DEVICE: DEVICE takes two arguments, parallel and serial. You use it to indicate the port to which your printer is connected. If absent, the default setting (parallel) is used.

FILE: The File Tool Type lets you indicate where and under what name you wish to store the file redirected by CMD. The default setting is FILE=RAM:CMD_file. If you use CMD to save more than one file, it will not name the files sequentially: The second file will be CMD_file.1, the third CMD_file.2, and so on.

SKIP: This is a Boolean Tool Type. You use it to indicate whether CMD should skip any initial write that it intercepts. Such initial writes are often printer initialization codes that are not strictly a part of the file. The default setting is SKIP=True, causing CMD to ignore initial writes.

MULTIPLE: Another Boolean Tool Type, MULTIPLE lets you determine whether CMD should stay active for an indefinite period to redirect multiple files or whether it should remove itself after it has redirected one file. The latter case (MULTIPLE=False) is the default. After choosing the MULTIPLE=True option, close down CMD by double-clicking on its icon again.

NOTIFY: Also a Boolean Tool Type, NOTIFY lets you specify whether CMD should keep you appraised of its progress by issuing the messages I mentioned above. The default is True.

Colors

If the Colors tool gives you a feeling of *deja vu,* you're not alone. When you click on the Colors icon, the tool opens a window (see Figure 6-2) that looks remarkably like that of the Palette Preferences editor. Unlike the colors you set with the editor, the colors you choose with Colors will be lost when you

reboot. The other difference is that Colors uses hexadecimal numbers (base 16), to indicate the different red, green, and blue values, while Palette uses decimal numbers. The former may be of greater use to programmers who are experimenting with different colors for their programs. The big plus of Colors is that it can change the colors of *any* Amiga screen, not only the Workbench screen.

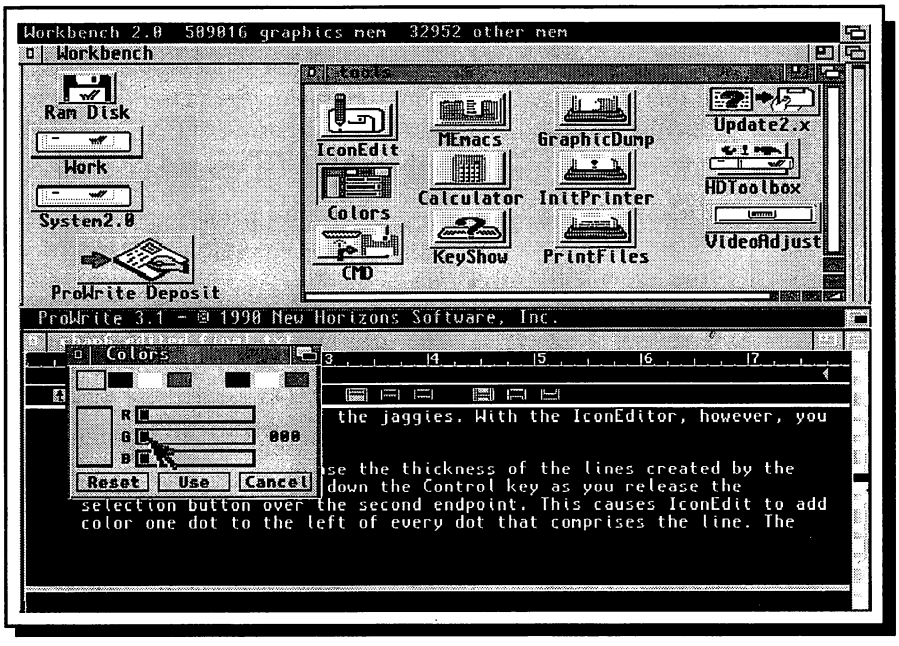


Figure 6-2 The Color Program

When you open Color with another screen in the foreground, it opens on that screen, allowing you to change the palette of the screen. In the figure, Color is open on an eight-color ProWrite screen.

To use Colors with a screen other than Workbench, you first need to move the custom screen to the front with the screen-depth gadgets or the IHelp CYCLESCREEN function. Next, drag the custom screen about halfway down the display to uncover the Workbench screen. Now, move the pointer to the Workbench screen and open the Colors window, which has the remarkable property of always appearing on the topmost screen. The Colors window displays the current palette of the screen on which it opens, adjusting the number of colors it displays to match the properties of the screen. The Reset button returns the palette to the condition that existed when you opened Colors. Use causes the screen to adopt the palette you've set, while Cancel scraps the changes.

GraphicDump

The GraphicDump tool prints the top-most screen on your display with the printer you select using the Printer editor. When you open GraphicDump, it loads itself and then gives you about 10 seconds to move the screen you want to the front and to arrange its windows, menus, and icons to your liking. The only thing that GraphicDump can't print is the pointer. Pointers are sprites, they are not a part of the underlying display.

Because GraphicDump uses the Preferences-set printer driver, the printout uses the parameters you set with the PrintGfx editor. GraphicDump does support one Tool Type, SIZE, that affects the printout; however, it is only effective if the Limits Type gadget in the PrintGfx editor is set to Ignore. When active, SIZE uses one of the following parameters.

SIZE=tiny: This parameter prints the screen with a horizontal dimension 1/4 the maximum allowed by the printer. As with all the modes, the height is automatically set to maintain the proper aspect ratio.

SIZE=small: Prints the image at 1/2 the allowable maximum width.

SIZE=medium: Prints an image at 3/4 the allowable maximum width.

SIZE=large: For a full-width picture, choose large.

SIZE=<xdots>:<ydots>: This parameter lets you specify absolute values for the number of dots that will comprise the horizontal <xdots> and vertical <ydots> resolutions of the printout. These values are subject to the capabilities of the printer.

IconEdit

A large application that creates its own menu bar and features a complex input window, IconEdit lets you create and alter icons. This is not simply an aesthetic function; IconEdit is indispensable for creating icons for tools and projects that don't have any.

When you open IconEdit, you see a window like that in Figure 6-3. A magnified view of the default tool icon fills the left two-thirds of the window. This view is the canvas of a miniature paint program where you create new icons. Beside the magnified view are the color gadget, six painting tools, and Clear and Undo buttons. The last column of the window contains normal-sized views of the icon as it will appear both deselected and selected, a button to toggle the magnified view between these two views, and a positioning gadget.

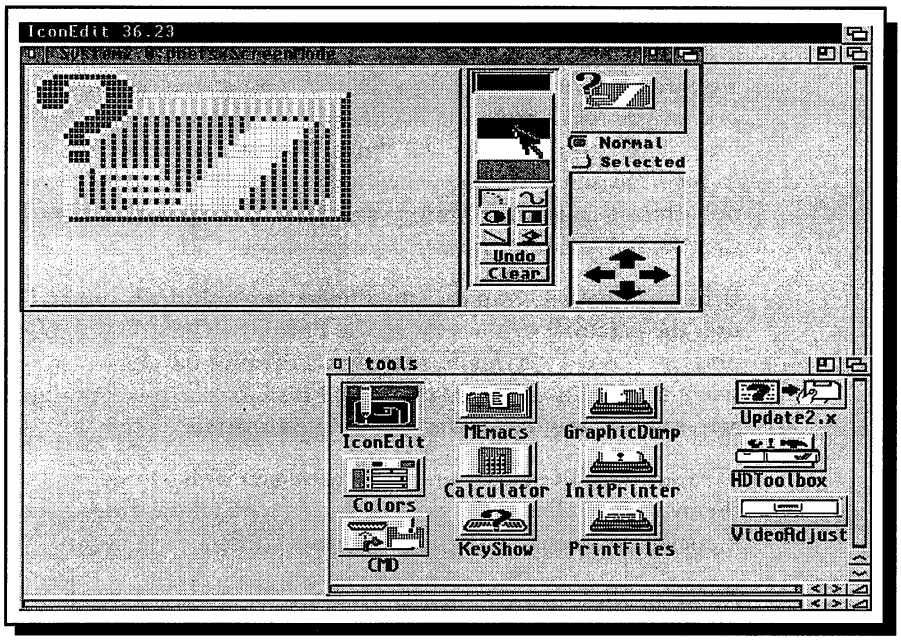


Figure 6-3 IconEdit

Most of the IconEdit window is taken up with the icon display. Here, the display contains the image of the Workbench default tool icon.

Magnified View: This box, like the magnified view areas in the WBPattern and Pointer editors in the Prefs drawer, is an active drawing area that lets you change the color of any point using the mouse pointer. You select the drawing color from the color gadget to the right of the view area. To draw in the magnified view area, move the pointer to where you want to draw and press the select button to change the color of the spot beneath the pointer. If you drag the mouse pointer slowly, it will change the color of all the points that appear beneath it.

The magnified view always contains the image of the Workbench default tool icon when you open IconEdit. Notice that the icon doesn't fill the view. You can create icons larger (or smaller) than the default icons.

The size of the icon you create with IconEdit is limited by the size of the magnified view area. Inside this area, the size of the icon is determined by the background color, which is the first color in the color gadget under the selected color box. IconEdit uses the background color to determine where to crop icons that it creates. When it saves an icon, IconEdit doesn't save the entire

magnified view; rather, it saves a rectangle that begins in the upper-left corner of the magnified view and contains all the dots in the view that are not the background color, plus a four-or-five-dot border around this area.

Color Gadget: This gadget consists of a number of colored rectangles that reflect the number of colors you set in the ScreenMode editor and the color values you set with the Palette editor. The box at the top of the gadget displays the currently selected color. To change it, move the pointer over one of the other colors and click the left mouse button.

Because you can have Workbench screens that display up to 16 colors, you can create 16-color icons with IconEdit. If you later switch your Workbench back to eight, four, or even two colors, however, the extra colors are lost.

The IconEdit color gadget has the interesting ability to let you select two colors at a time. First, you select one color normally; then, shift-select another color. In the selected color box, you'll see a color that looks like a mix of the two you've selected. It is actually a checkerboard pattern of alternating dots of the two colors. This hybrid color isn't used when you draw simple points and lines; these use the first of the two colors you selected. The hybrid colors are instead used in the area-painting gadgets — Circle, Box, and Fill — that are discussed below.

Painting Gadgets: Below the color gadget are six small buttons with different shapes drawn on them. These are the painting gadgets: Freehand, Continuous Freehand, Circle, Box, Line, and Fill. The Painting gadgets act like radio buttons; when you select one, you deselect another. One painting gadget is always active.

Freehand Gadget: Active when you first open the IconEdit window, the Freehand gadget changes the color of the dot below the pointer whenever the select button is depressed. If you drag the mouse quickly, however, the gadget may not be able to color all the intervening dots between the old position of the pointer and the new one. The priority of the freehand gadget is to keep up with the pointer; it will skip intervening dots to do so.

Continuous Freehand Gadget: This gadget differs in one significant respect from the freehand gadget. Its priority is not to keep up with the pointer, but to change the color of every dot you drag the pointer over. Thus, it may lag behind the pointer if you drag the mouse quickly, but it will not miss any intervening points.

Circle Gadget: One side of the imagery on this gadget, shows a hollow circle and the other shows a filled one. By selecting one side or the other, you can choose to draw either the perimeter of a circle or a completely filled one. The procedure for actually drawing hollow and filled circles is the same.

After selecting the Circle gadget, move the pointer to the pixel in the magnified view that you want to be the center of the circle. Press the select button and hold it down. Now, drag the mouse on a horizontal line away from the center; you should see a straight line forming on both sides of the center. Next, drag the mouse straight up or down and you'll see the line opening into an oval and then a circle. Dragging the mouse in different directions lets you change the shape of the circle at will. When the circle is in the shape you want, release the selection button. If you draw a hollow circle, the shape you just created remains in the view window; if you use the filled-circle, the new shape is immediately filled with the currently selected color.

If you used a hybrid color from the Color gadget, the filled circle will consist of alternating dots of the two colors. Note that in the currently-selected color box and the normal view box, this checkerboard pattern often looks like another color intermediate between the two that make up the pattern. This is an excellent example of using dithering to create the illusion that you have more colors available than you actually do.

If you use the hollow-circle option, you can double the thickness of the left and right sides of the circle by pressing and holding the Control key as you release the selection button. Experiment with this option to see its effect.

Box Gadget: The Box gadget works similarly to the Circle gadget. Select the left side of the gadget to draw the outline of a box; select the right side for a filled box. This time, however, the dot the mouse pointer is over when you first press the selection button becomes one of the box's corners. Which it is depends on where you drag the pointer to before you release the button. Like the Circle gadget, the filled-box option takes advantage of shift-selected colors, and the hollow-box option doubles the right and left sides of the box via the Control key.

Line Gadget: As you might expect, the Line gadget lets you draw straight lines quickly and easily. With it selected, you move the pointer to the magnified view area and press the selection button over the point you want as one endpoint. Now, drag the pointer to the other endpoint and release the button. That's all there is to creating a line.

You'll notice, however, that not all lines on the Amiga — or on any computer, for that matter — are created equal. Except for horizontal and vertical lines, and for those diagonal lines that match the slope of the Amiga's aspect ratio, all straight lines exhibit the jaggies to one degree or another. The term jaggies describes the staircase-like look exhibited by most so-called straight lines on a computer. Sophisticated programs can use small color gradations to minimize the effects of the jaggies. With the IconEditor, however, you have to live with them.

You can increase the thickness of the lines created by the line gadget by holding down the Control key as you release the selection button over the second endpoint. This causes IconEdit to add color one dot to the left of every dot that comprises the line. The technique doesn't have much affect on the jaggies, but it makes lines look more solid in the normal-sized viewing modes.

Fill Gadget: With the Fill gadget selected, you can fill any bounded area in the magnified view with the current color by selecting any point within the bounded area. A bounded area is an area in the magnified view that is completely surrounded by dots of one color. For example, when you create a circle with the Circle gadget, you create a bounded area inside the circle. You can use the Fill gadget to change the color of every dot inside the circle.

To put it more precisely, the Fill gadget changes the color of the dot beneath the pointer to the selected color. It then changes the color of every dot that is both the same color as the original dot and contiguous to it in a cardinal direction (up, down, left, or right). It then repeats the procedure, treating every dot that changed color as it did the original dot until there are no more dots that fulfill both conditions of change. Besides a mouthful, that is a description of a recursive algorithm. To get a feel for how the Fill gadget works, you have to use it. Like Circle and Box, the Fill gadget takes advantage of the hybrid colors created by shift-selecting a second color in the Color gadget.

Clear: Choosing Clear paints the entire magnified view with the currently selected color.

Undo: Selecting the Undo button restores the magnified view to the condition it was in before you last pressed the selection button inside the view window or over the Clear button. Practically speaking, it erases the last action you took in the magnified view window. For example, if you hit the Clear button by mistake and wiped out your drawing, Undo restores it. Note that you can't reverse an Undo.

Normal View: In the upper-right corner of the IconEdit window is a standard-sized view of the icon you're working on in the magnified view. This is what the icon will look like when it is deselected. Note that if you hold the selection button down on this view, you can see how the icon will look when selected.

Normal/Selected Button: Below the normal view is a radio button that lets you switch what you see in the magnified view window. With the Normal button selected, the magnified view lets you edit the icon's deselected image. Click on the Selected button to move the selected version of the icon to the magnified view window. Note that this button is only active when you choose the Image item from the Highlight menu.

Selected View: With the Image item active, this box will show the image that will be displayed when the icon is selected. You can edit this image in the magnified view box by hitting the Selected button.

Positioning Gadget: Below the Selected View box are four arrows that let you move the dots that make up the magnified view box. Selecting an arrow moves the dots in the indicated direction. If you move part of the icon image beyond the borders of the view box, the image gets clipped by the border. You can't retrieve anything you've moved beyond the borders of the view area. One annoying thing about the positioning arrows is that you can't hold the selection button down to make them repeat. You must continuously press and release the button.

IconEdit Menus

IconEdit uses five menus (Project, Type, Highlight, Images, and Misc) in conjunction with the gadgets in its input window.

Project Menu: The six items in the Project menu let you load old icons and save new and edited ones.

New: Unlike in most Amiga programs, here the New item does more than simply erase whatever you have in the magnified view area. New also loads one of the system default icons into the magnified view for editing or saving. You determine which icon will appear in the magnified view when New is chosen by accessing the Type menu (see below). The default icon of the desired type appears in the magnified view when you select the New item. If you access New after changing the image in the magnified view or the type of icon using the Type menu, you'll see a requester asking whether you want to cancel the New operation, save the current icon before creating a new one, or continue with New. Click on the appropriate button in response.

Open: Accessing this item brings up a standard file requester that lets you open any icon file on your system. You can open an icon by either selecting a .info file directly or by selecting the file associated with a .info file. For example, choosing Calculator from the file list will have the same effect as picking Calculator.info. If a file is not an icon file and does not have an icon associated with it, the Open command will tell you so. When you load an icon, you get more than its image. The menu selections in the Type and Highlight menus also change to match the characteristics of the newly loaded icon.

Save: Save overwrites the icon file you previously loaded using the Open item with the contents of the magnified view window. If you have not previously loaded an existing icon or you selected New in the meantime, IconEdit considers Save to be Save As.

Save As: Unlike Save, which automatically overwrites a known file, Save As brings up the Amiga file requester and lets you either enter a name for the icon file or choose a file you want to overwrite. If you neglect to add the .info extension after the name of the icon, IconEdit does it for you. Be very careful when saving icons using Save As. You don't want to overwrite an icon file with one of another type. For example, if you save a drawer icon with the name Calculator to the Tools drawer, you'll be unable to open the Calculator program from Workbench. Use this item with care.

Save As Default Icon: This item lets you save the new icon as one of the default system icons. Which default icon it becomes depends upon the current icon type you indicated in the Type menu. When you save an icon as a default, IconEdit saves it to both the Env:Sys drawer and the Envarc:Sys drawer. Normally, these drawers are assigned to RAM:Env/Sys and Sys:Prefs/Env-Archive/Sys, respectively. When Workbench needs to use a default icon (for example when you choose Show All or access the New Drawer item) it checks Env:Sys to see if you saved an icon file that overrides the built-in defaults. If you have, Workbench uses that file instead of its built-in icons.

If you save an icon as a default icon, it takes its filename from the type of icon it is. For example, the tool default icon file is called def_tool, and the drawer default is called def_drawer. Because these files are stored permanently on disk, they survive when you reboot your machine. To revert back to the built-in Workbench defaults, you must load these from your system disk and reboot.

Quit: Choosing this item is the equivalent of selecting the close gadget on the IconEdit window. If you haven't saved the current image in the magnified view area, IconEdit will put up a requester asking if you want to do so. The Save First button lets you save the icon before quitting, while Continue quits without saving. The Cancel button aborts the quit operation.

Type Menu: The items in the Type menu (Disk, Drawer, Tool, Project, and Garbage) correspond to the five types of icons you have available on your system. When you're working with an icon, you can determine its type by simply choosing the type you want from the menu. When you save an icon, it takes its type from the current setting of the Type menu. The five items in the Type menu are mutually exclusive; when you select a type, you automatically deselect all the other types.

Highlight Menu: The items in this menu let you indicate how you want an icon to appear when it is selected. They are mutually exclusive; an icon cannot use multiple methods to show that it is selected.

Complement: When an icon uses the complement method, it shows that it has been selected by swapping its original colors with their complementary colors. The colors on a standard four-color Workbench are numbered from 0 to 3. For example, the background color — gray, if you haven't changed the default palette settings — is color 0. In fact, the colors are shown in the Color gadget in their numerical order. When you select an icon that uses the complement method to show selection, it swaps its color 0 for the highest numbered color in its list, color 1 for the next highest numbered color, and so on. Deselecting the icon reverts back to the original colors.

Backfill: New to Amiga OS 2.0, this method of indicating selection helps give Amiga icons their 3-D look. Workbench 2.0 icons are supposed to look as though they are raised above the plane of the Workbench. When selected, an icon is supposed to look as though you had pushed it below the plane of the Workbench. Commodore swaps the position of the thin white and black lines — denoting light and shadow — that surround every icon to achieve this effect. All icons on Workbench 2.0, even those created before 2.0 existed, exhibit this 3-D effect.

To enhance the 3-D effect, Backfill complements the colors in the icon just as Complement does except when the border of the icon matches the background color of the screen. In this case, it fills the area from the borders of the icon — using the same algorithm as the Fill gadget — with the background color of the screen. Thus, if the background color of your icon does not match the Workbench background color, a Backfill icon behaves like a Complement icon. If the backgrounds do match, then the border area of a Backfill icon is not complemented when it is selected.

Image: The most interesting icons are those that display a completely different image when they are selected. The Prefs drawer is an excellent example of this, although, if you look closely, you'll see that all the Workbench drawers use an alternate image when they are selected. When you access the Image item, you activate the Normal/Selected button on the IconEdit window. You can now switch the magnified view to create and edit both the selected and normal images of the icon. Icons that use an alternate image store both images in the same .info file.

Images Menu: This menu contains items concerned with creating icons that display an alternate image when selected. Using any of these menu items, automatically resets the selection in the Highlight menu to Image.

Exchange: Choosing this item swaps the image in the Normal view box with what is in the Selected view box.

Copy: If the Normal/Selected button is on Normal, then this item copies the image in the Normal view into the Selected view and switches the button to Selected. With the Normal/Selected button set to selected, Copy makes the opposite switch.

Load: This item consists of a four-item submenu.

Load Normal Image: Via the standard file requester, this item lets you load the normal image — and only the image — of any icon on the disk. You can load the image into either the Normal or the Selected view boxes, depending upon the state of the Normal/Selected button.

Load Selected Image: This item lets you load an icon's selected image into either the Normal or Selected view boxes, depending upon the condition of the Normal/Selected button. If you try to load an icon that does not show an alternate image when selected, you get a message saying "No Image."

Load Both Images: You access this menu item when you want to load both the normal and selected icon images from a disk file. After you select a file from the requester, IconEdit attempts to load both icon images into the appropriate view windows. If the file doesn't have a separate selected image, you'll see a "No Image" message.

Load IFF Brush: Many Amiga paint programs offer the capability to paint using software "brushes." IconEdit can load these brushes into both the Normal and Selected view boxes. Thus, you can use the sophisticated features of a paint program to create your icons, and simply use IconEdit to merge an image with an icon type. If you select anything other than an IFF brush file from the item's file requester, you'll get an error message saying that the file isn't an IFF file. After you've selected a brush file, the program will load it into the view window indicated by the Normal/Selected button. Note that the magnified view area is approximately 80 pixels wide and 40 pixels high. If you load an IFF brush that is larger than this, IconEdit will crop it along the right side and bottom to fit in the view.

Save IFF Brush: This option lets you save the image in either the Normal or Selected view boxes as an IFF brush. Given the better graphics editing features of an IFF paint program, you won't do this often.

Restore: Accessing this item restores the image in view that was present when you opened the IconEdit window to the magnified view. Note, however, that any changes you made in the Type and Highlight menus are still in effect.

Misc Menu: A grab bag, the Misc menu has three functions.

Grid: With this option activated, the dots in the magnified view window are separated into individual cells by a tiny grid of vertical and horizontal lines rendered in the background color for easier access. Switching Grid off removes the lines and lets the dots touch adjacent dots.

Remap B/W: This item is well named only if you use the default Workbench colors. When selected, it turns all the color 1 dots to color 2 and all the color 2 dots to color 1. The item gets its name from the fact that black is color 1 and white is color 2 in the default Workbench palette.

Auto TopLeft: The final choice moves the image in the magnified view window so that the left-most, non-background color dot in the image touches the left border of the box and the top-most, non-background color dot touches the top border.

IconEdit: A Practical Example

More than simply a place to create pretty pictures, the IconEditor can also provide files with icons, a valuable function. For example, using the AmigaDOS text editor, Ed, I created a file that contained the names of some picture files I wanted the Display tool (see Chapter 5) to display. I saved the file to the Utilities drawer of my System2.0 disk, giving it the name PictureList, and then exited the text editor. Because Ed does not create .info files, PictureList did not show up in the Utilities drawer when I opened it. Worse, because it didn't have an icon, I couldn't enter the Tool Type that would tell Display that PictureList was a list of picture files. IconEdit solves the problem.

When I opened IconEdit, it came up with the image of the default tool icon. I wanted the image of a project icon, so I chose the Project item from the Type menu and selected New from the Project menu. The resulting requester informed me that "<untitled> had been changed." I selected the requester's Continue button, as I didn't care about the tool icon image I was erasing. The program then loaded the default project icon (see Figure 6-4). Rather than trusting my artistic talents and editing the image, I simply accessed the Save As item from the Project menu.

Save As brought up the standard Amiga file requester. I wanted to get to the Utilities directory, so I first clicked the Disks button and then selected System2.0 from the list of drives. As the requester listed the files and drawers on the System2.0 disk, I clicked once in the scroll gadget to request an alphabetized display with the drawers below the files. I then scrolled down to the bottom of the list, where I found and selected the Utilities drawer. Toward the bottom of the list of files in Utilities, I found PictureList. I selected it, so that it appeared in the File gadget of the requester, and then I clicked on OK. IconEdit appended a .info to the name PictureList, giving the file the name PictureList.info, and saved it to the Utilities drawer. Once it was in the same drawer as PictureList, it became that file's icon file.

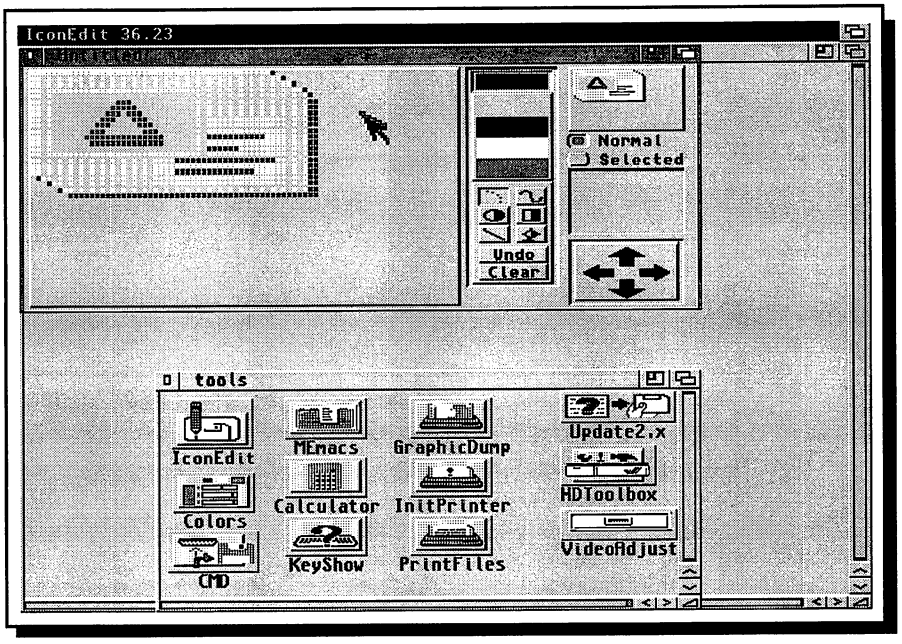


Figure 6-4 Creating a Project Icon

By selecting Project from the Type menu and then New from the Project menu, you get the default project icon. You can then save this icon as a .info file.

Because my file list now had an icon, I was able to call up its Information window and enter Sys:Utilities/Display as its default tool and FileList=True into the Tool Type list. In doing so I created a project file for a tool that doesn't create its own. Have fun creating interesting icons with IconEdit, but don't forget that it serves another important function as well.

InitPrinter

Unlike IconEdit, InitPrinter is a simple program without a window or a menu bar. It takes the information you entered with the Printer and PrintGfx editors and passes it to the printer. Although most programs that use the printer also initialize it, InitPrinter is useful for those programs that don't.

KeyShow

Suppose you had to write a report that included some foreign phrases and needed foreign-language characters, such as an umlaut. Before you switch

keymaps to access these characters, think again. The usa1 character set, for example, contains many foreign-language character and symbols and KeyShow shows you how to access them. (The discussion that follows assumes that you are using the usa1 keymap. Other keymaps have equivalent features to the ones described here.)

When you open KeyShow, it opens the window shown in Figure 6-5. The characters shown on the keycaps are what the keyboard sends to the Amiga when you press each key. The original KeyShow display shows the standard lowercase character set; I'll explain why some are shown in bold italic type and others are in plain text later.

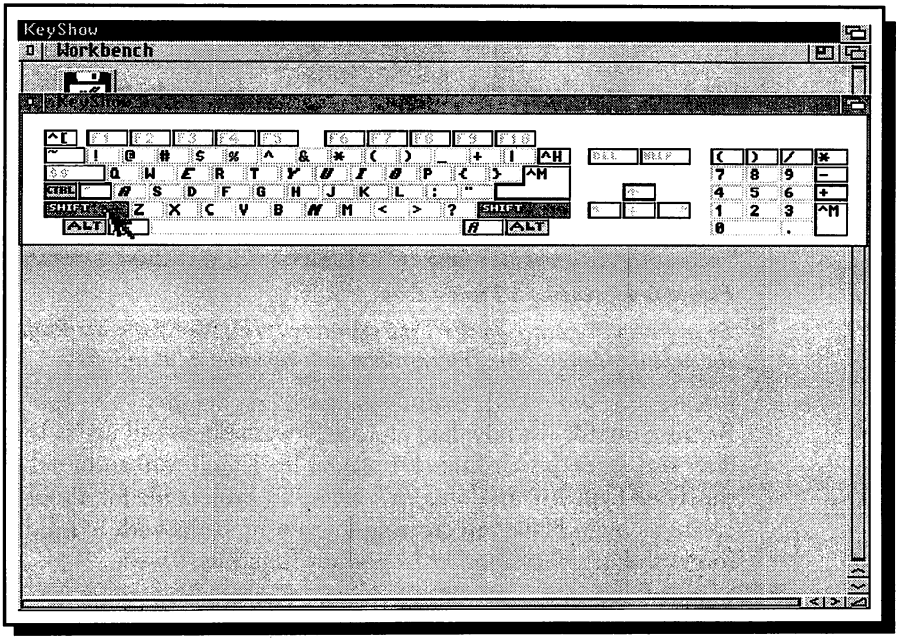


Figure 6-5 KeyShow Display

KeyShow lets you see the mappings of your keys. By clicking on the Control, Shift, and Alt keys, you can see how these keys alter the characters produced by the alphanumeric keys.

Press a Shift key on your keyboard, or select a Shift key on the KeyShow display. The Shift key turns blue, and the keys display their shifted characters — capital letters for the alphabetic characters and the special symbols for the numeric characters. The Shift keys, along with the Control key and the Alt keys, are called qualifier or modifier keys. They don't produce characters themselves, but, when pressed in conjunction with another key, they modify its character.

Now select the Control key or simply press it on your keyboard. You'll see all the control characters you can send from the keyboard. Preceded by a tilde (~) or a caret (^), control characters are nonprinting characters that convey special information. Because the ASCII code computers use was originally developed for teletype communications, most of the control codes deal with page-formatting information. Control-M, for example, is a carriage return; Control-I is a tab.

As you saw with the Display tool, some Amiga programs let you use Control characters to send instructions to the program. The most common control character is Control-C, which many programs use as an alternate way to break out of the program.

The most interesting special characters are produced in conjunction with the Alt keys. Both shifted and unshifted, the Alt keys give you access to foreign characters and accents. Alt-L and Alt-l for example, produce the symbol for the British pound (£), while Alt-s produces the Greek beta (ß) and both Alt-u and Alt-U give you the scientific symbol for a micron (µ). The Alt keys also let you place accent marks over standard characters. You can accent any character that appears on the screen keyboard in bold-italic type. Thus, you can accent an A, but not an S; an N, but not an M. To accent a character, you first type one of the accent keys; Alt-f (gauche), Alt-g (grave), Alt-h (circumflex), Alt-j (tilde), or Alt-k (umlaut) — the shifted characters work as well — and then type the character you want to accent. For example, to put an umlaut over an e, you press Alt-k, followed by e.

With the usa1 keymap, all the qualifiers have strange effects on the Tab key. The Control and Alt keys produce a blank on the Tab key display, and Shift-Tab results in When used in a word processor, both Control-Tab and Alt-Tab produce a tab, while Shift-Tab flashes the screen of the word processor's screen. Control- and Alt-Tab default to the function of an unmodified Tab; Shift-Tab sends an unidentified character code to the Amiga.

MicroEMACS

Shorthand for MicroEMACS, MEmacs is the filename of a screen-oriented, multiple-file text editor. Based on EMACS, a text editor originally developed on minicomputers, MicroEMACS shows its roots. It is not a true Amiga Intuition program, but a command-line program with a large and confusing menu system grafted onto it. For simple jobs like creating small AmigaDOS scripts or FileList projects for the Display tool, MicroEMACS is overkill; I prefer the simpler Ed editor. For bigger jobs, such as creating C program source files — the job EMACS was created to do — MicroEMACS is an acceptable tool. You

might want to check out some commercially available text editors, however, before you commit to producing a quarter-million lines of code with MicroEMACS. Like any text editor, MicroEMACS lets you enter and edit text, save files to disk, and load previously saved files. You can think of a text editor as a word processor without the fancy formatting and printing capabilities.

When you run MicroEMACS, it opens a two-color custom screen. If you don't like its black on gray color scheme, use the Color tool to change it. The screen consists of a title/menu bar and a large, text buffer area with the cursor (called the dot in MicroEMACS) in the upper-left corner. At the bottom of the buffer is a line that identifies the program (MicroEMACS), the buffer (main), and, if you've loaded a file into the buffer or saved the buffer to disk, the name of the file associated with the buffer. Below this status line is a command line where you enter the arguments for the menu commands.

To enter text into the buffer, simply start typing. When you get to the end of the line, you'll notice that, unlike a word processor, MicroEMACS doesn't automatically break the line and wrap it to the next line. It simply puts a $ at the end of the line and lets you continue entering text, although you can't see any text beyond the end of the line. You don't move to a new line in MicroEMACS until you press Return. These features, more than anything else, point up MicroEMACS roots as a programmer's text editor. If you want to change the program and put it into wrap mode, you have to access the Set item in the Extras menu.

The dot marks the point at which you enter text. It moves to keep up with the text you're entering. You can move the dot, and therefore the point at which you enter text, by clicking on a new spot with the left mouse button or pressing the keyboard's cursor keys. When you move the dot to the midst of a line of text and begin typing, MicroEMACS inserts the new text by displacing the existing text to the right. The Del key erases the character under the dot, while Backspace deletes the character to the dot's left.

MEmacs File Handling

MicroEMACS lets you manage files through the Project menu. Once you've entered text into the main buffer, you can save the file to disk in several ways. The most obvious method, choosing the Save-File item, turns out to be wrong. You have to name a file before you can save it. You choose the Rename item, enter a name for the file on the command line, then access Save-File. For quicker results, you can choose the Save-File-As item, which prompts you for a filename and then saves the file, all in one stroke. Once you've named a file, you can use Save-File to save it from then on. MicroEMACS has two other

Save options: Save-Mod saves all buffers that you've modified since you last saved them (as you'll see, MicroEMACS lets you open multiple buffers), and Save-Exit saves all the modified buffers and then exits the program.

Saving a simple file with MicroEMACS teaches two important aspects of the program. First, every menu item has a keyboard equivalent. Second, MicroEMACS does not use the standard Amiga file requester: It doesn't use any file requester at all. You have to understand AmigaDOS pathnames to use the program.

You'll especially miss a file requester when you try to load a file into MicroEMACS, because you must know the path and name of the file at the outset. The most frequently used file input command is the Read-File item. When you access this item, MicroEMACS prompts you to enter the name of the file you want to load on the command line. Once you've entered the pathname of the file, MicroEMACS loads it into the main buffer, overwriting whatever was there before. Two other commands load files into MicroEMACS without destroying the main buffer's contents. Visit-File lets you load a file into another buffer, which takes the name of the file you loaded. Insert-File lets you load a file into the current buffer on the line above the position of the dot.

To compensate for not using file requesters, MicroEMACS provides for two ways to access AmigaDOS commands from the Project menu. The first, New-CLI, opens a Shell on the Workbench screen. You can then enter all the AmigaDOS commands you wish (such as Dir to see a list of files and directories); EndShell or EndCLI closes the window and sends you back to MicroEMACS. Note that if you use the screen depth gadgets to move back to MicroEMACS while a spawned CLI is open, you won't be able to use the program; it is locked until the CLI process ends. The second way to access AmigaDOS commands is through the CLI-command item, which lets you enter a single AmigaDOS command on the MicroEMACS command line and puts the output of the command into a buffer called spawn.output that you can access and edit. Whenever you use the CLI-command item, the output of the latest command overwrites the previous contents of spawn.output.

There are two other items in the Project menu. Quit exits the program, after asking you if you want to save any buffers that have been modified since the last time they were saved. The About item lists the authors of the program.

More on Buffers

As you've seen, you can load different files into multiple MicroEMACS buffers. The Edit menu lets you move among the buffers and move blocks of text between buffers or between different parts of the same buffer.

If two buffers are visible on screen (see Figure 6-6), you can move from one to the other by clicking in the destination buffer. If the buffer you want to work with is not currently visible, you can move to it by choosing the Select-buffer item from the Edit menu and then entering its name on the command line. If you don't know which buffers are present, the List-buffers item will show you.

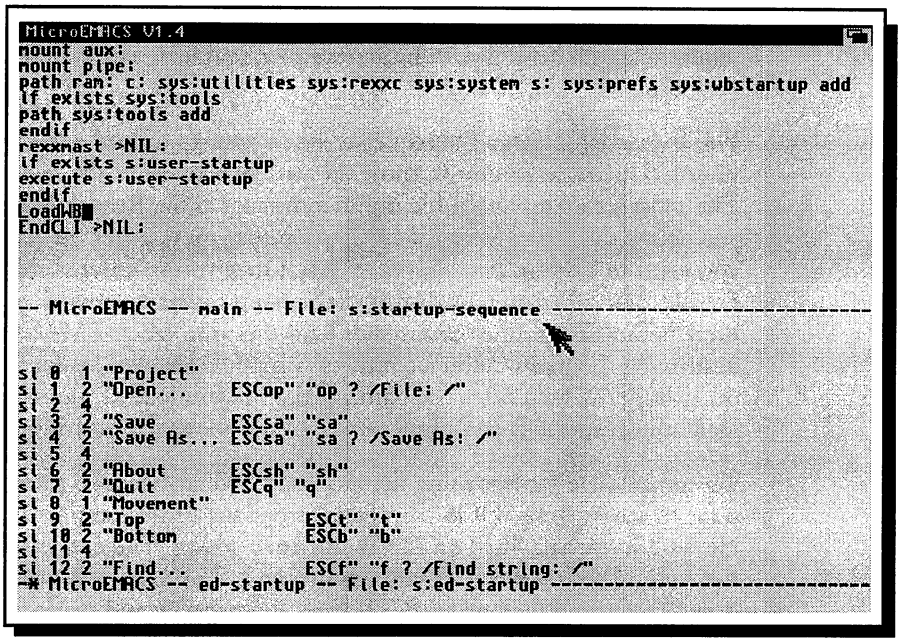


Figure 6-6 MicroEMACS Display

MicroEMACS lets you divide its display between multiple buffers. You can view different parts of the same file, or different files, in the separate buffers.

All text editors perform cut-and-paste operations, and MicroEMACS is no exception. It lets you select a block of text you want to cut or copy from a buffer, and then lets you paste that text back into the original buffer or into another. A block of text can be a character, word, sentence, paragraph, entire document, or any part of the text in the buffer. In MicroEMACS nomenclature, the cut operation is referred to as kill, paste is yank, and copy, mercifully, is copy.

Before you can kill or copy a block of text, you must select the block. You do so by marking the beginning of the block and then moving the dot to the end of the block. To mark the beginning, move the dot there and then either select Edit's Set-mark item or double-click on the dot. Now, simply move the dot to define the end of the block.

To delete the block, you select the Kill-region item, which cuts the text from the buffer and places it in a special kill buffer. If you kill successive regions of a buffer, each is appended to the buffer as long as you haven't yanked anything from the kill buffer or copied anything into it.

To copy the contents of the kill buffer into a normal buffer (a paste operation), move the dot to the point where you want the text inserted and choose the Yank item. You can thus move text from one buffer to another by killing it in one and yanking in into another.

You can also move text into the kill buffer without deleting it from its original buffer: Choose the Copy-region item instead of Kill-region. Another way to move text from one buffer to another is by positioning the dot at the text's new location and then choosing Insert-buffer. MicroEMACS prompts you for the name of the buffer you want, then inserts the text on the line above the dot's current position. Insert-buffer is a copy operation; it doesn't delete the contents of the inserted buffer.

Other capabilities available from the Edit menu are items that let you set regions of text to all uppercase (Upper-region) or all lowercase (Lower-region), an item that kills the entire contents of a buffer (Kill-buffer), and one that left-justifies a buffer's contents (Justify-buffer). You can also redraw the screen (Redisplay), use MicroEMACS command strings in the test (Quote-char), have the dot autoindent (Indent), transpose the position of two characters (Transpose), and cancel an ongoing operation (Cancel).

Windows Plus

One of the more interesting features of MicroEMACS is its use of windows: You can split a buffer into multiple windows. Each window can move to any part of the buffer, and the changes made in any one window are reflected in all other windows open to the same buffer. MicroEMACS windows are nothing like Workbench windows; rather, they are views into a buffer.

The Window menu contains items that let you split a buffer into two windows, and then split these window further (Split-window), that let you recombine all the windows open on a buffer into one window (One-window), and that let you move around among windows (Next-window and Prev-window). You can also change the size of windows (Expand-window, Shrink-window), and scroll by page inside a window (Next-w-page, Prev-w-page). Windows are great when you have to see the contents of one part of a file while entering text or editing another part.

The level of detail in the next three menus give you an idea of the number of ways that MicroEMACS gives you to manipulate text. The Move menu contains items that let you move the dot about a word, line, page, window, or even a buffer at a time. Next to Move is the Line menu, which lets you position the dot on a line, delete a line, and delete part of a line. Following the Line menu is the Word menu, which lets you delete individual words and change the case of letters in a word.

The Search menu provides items that let you search forward and backward for a specific string of characters (Search-forward and Search-backward), both automatic and manual string replacement functions (Search-replace, Query-s-r), and Fence-match, an item that moves the dot to the next occurrence of the character currently under the dot.

The final MicroEMACS menu, Extras, lets you define how MicroEMACS performs some of its operations. The Set-arg item lets you specify how many times the operation following will occur. For example, if you wanted to type 15 r's, you'd access Set-arg, enter 15 in the command line, and then enter the letter r. Warning: Do not press the Return key after entering the value for arg, otherwise you'll wind up with 15 carriage returns.

The Set item lets you control a number of important program parameters. You can toggle MicroEMACS between its own screen and the Workbench screen by entering the word Screen, turn interlace on and off by entering Interlace, set the Left and Right margins, define the number of spaces for a Tab, define the number of spaces for an Indent, toggle between case-sensitive and case-insensitive searches (Case), and toggle the backup function by entering the keyword Backup and then one of the following: On, which backs the current file up to the T: directory, Safe, which keeps you from overwriting an existing file, and Off, which does none of the above. Most importantly, the Set item lets you enable word wrap at the end of a line. You first access Set and then enter wrap in the command line. You are then prompted for the column number where word wrap begins. With word wrap enabled, MicroEMACS is a much more tolerable general text editor.

The last few items in the Extras menu let you record, begin, and end macros, redefine just about any key on your keyboard as a macro key, and execute files of macro commands.

I've been deliberately short in my descriptions of many of the features in all but the first three MicroEMACS menus. Rather than repeat Commodore's 30 pages of documentation, I wanted to give you simply the information you

need to get started working with MicroEMACS. MicroEMACS has enough features that you could spend months learning them all, yet you can learn the ones you *need* in just a few hours.

PrintFiles

Because neither MicroEMACS nor Ed can print a file, Commodore provides PrintFiles with Amiga OS 2.0. PrintFiles sends the files you've selected to your printer and is used. It is used primarily to print ASCII text files produced by the Amiga text editors. You can print multiple files by shift-selecting the files and then double-clicking on the PrintFiles menu while the Shift key is down. To ensure that the output from each file in a multiple-file printout begins on a new page, you should add Flags=Formfeed to the Tool Types list of the PrintFiles Information window.

The VideoAdjust Drawer

One of the new features on the Amiga 3000 is its built-in Display Enhancer. The Display Enhancer eliminates the flicker that accompanies interlaced screens and also eliminates the visible scan lines on non-interlaced screens.

The Tools drawer of the System2.0 volume on Amiga 3000s contains a drawer called VideoAdjust. Inside you will find three icons: Test_1, Test_2, and Test_3. Test_1 and Test_2 are project icons: When you open one of them, it loads the Display tool in Sys:Utilities and then loads itself into Display. Although it looks like the other two, and its Information window says that it is a project, Test_3 is a self-contained program.

The test files help you adjust the output of your video display. Test_1 is a non-interlaced high-resolution screen that contains numerous patterns and gray scales (see Figure 6-7). With it on your display, you use the fine-tuning controls on your monitor to eliminate any flicker you see. Test_2 is a high-resolution interlaced screen of line patterns and color combinations. The idea once again is to use the fine-tuning controls on you monitor to produce as steady a display as possible. When you open Test_3, it opens a "Display Enhancer PLL Tuning Window" on your Workbench screen. By turning the Display Enhancer control on the back of your machine to produce the steadiest display in the tuning window, you ensure that you're getting the best possible output from the Display Enhancer hardware.

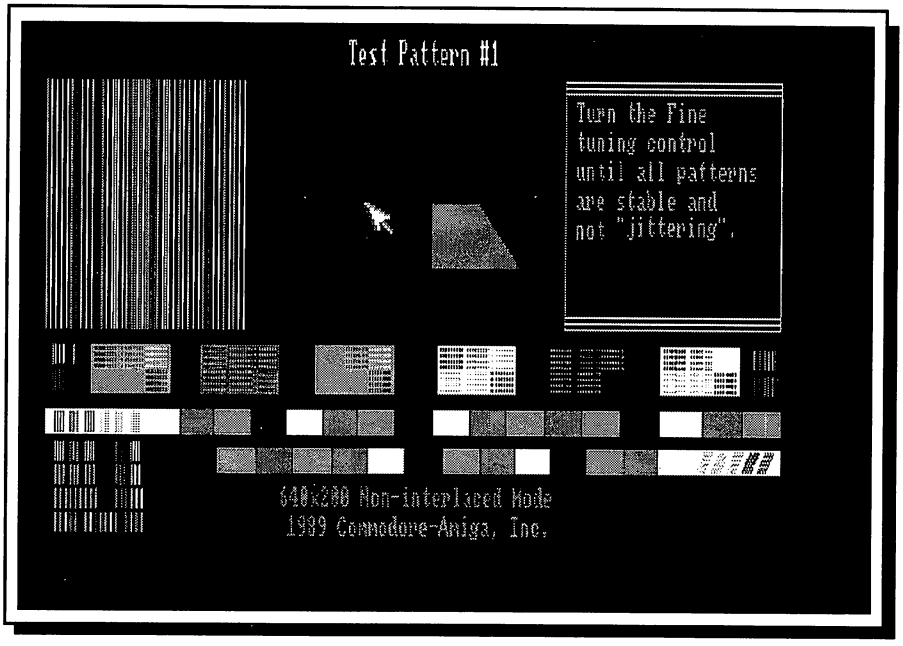


Figure 6-7 Tuning your Monitor

The three files in the VideoAdjust drawer give you test patterns you can use to adjust your monitor. This one is the first, Test_1.

HDToolBox

To many new Amiga owners, the setup, care, and management of a hard-disk drive is akin to travelling in the Twilight Zone: Everything looks familiar, but nothing works the same. The job of HDToolBox is to make setting up and using hard disks easier. HDToolBox lets you prepare a new hard disk for use or change the parameters of an existing hard drive. It is designed specifically to work with Commodore's hard-drive controllers on the Amiga 2000 and 3000, although it also works with the ST506 interface used in the A590 expansion box.

Hard drives are wonderful devices. Spinning at 3,600 rpm, they are much faster than floppies, and a lot roomier too. The problem is that you can't simply plug them in and start working. To use the drive correctly, the system must first receive details about the drive's physical characteristics. If the drive doesn't supply this information automatically, you must. Similar to a floppy, a hard drive has to undergo a low-level format that prepares it to receive data.

Then, you have to tell the system how you want the drive to be partitioned. Partitioning is the process of creating two or more volumes on a single hard-disk drive. Once the drive is partitioned, it then undergoes a quick AmigaDOS format which writes the appropriate root block information to the partitions. Finally, your drive is ready for use. HDToolBox tries to make preparation easier. (If you have a hard drive that is functioning normally and that is partitioned to your satisfaction, you don't have a need to access HDToolBox.)

When you open HDToolBox, the program first searches for any hard-disk drives attached to your system. Once HDToolBox has interrogated your drives, it displays the window shown in Figure 6-8. If you have a previously formatted hard drive, the manufacturer and model name of your drive will appear in the "Hard Drives in System" display. You can proceed directly to the section on partitioning a drive. If the drive is new and unformatted, it will show up as "Unknown" under the drive-type heading. In this case, you'll have to select the Change Drive Type button to supply the system with the information about your drive.

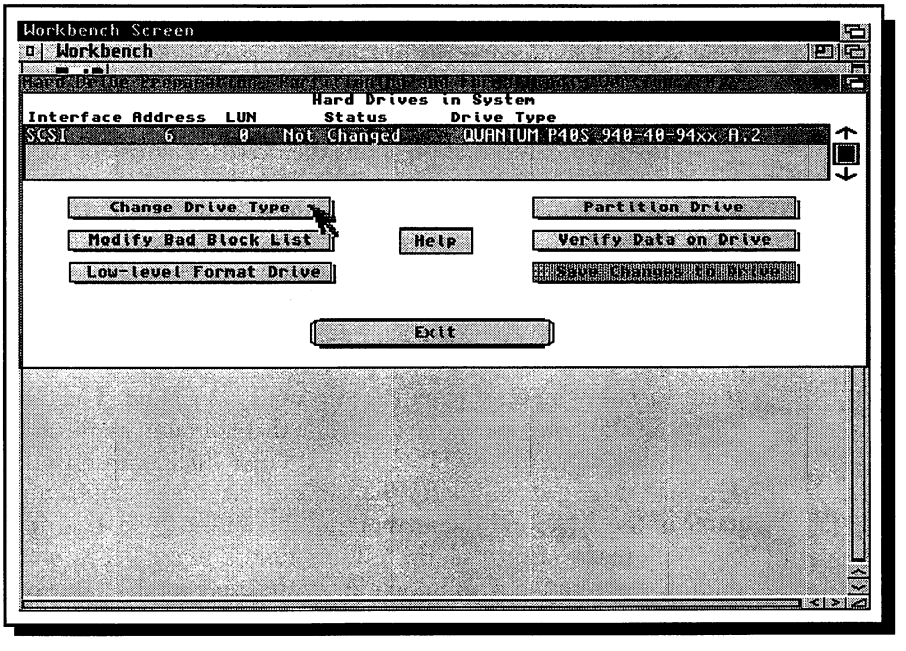


Figure 6-8 HDToolBox

The main window of the HDToolBox program shows the hard drives currently attached to your system. Using the buttons below, you can prepare the hard drive for use and partition the space on it between multiple volumes.

Change Drive Type

When you attach a new hard drive to your system, AmigaDOS needs to know the physical configuration of the drive (number of heads it has, number of cylinders, and so on) to format it properly. To enter this information about an Unknown drive, you select it from the drive list and then hit the Change Drive Type button.

In the resulting Set Drive Type window, you first select the type of interface you are using with the drive by clicking on either the SCSI or XT (ST506) buttons. Below these buttons is a list of drives whose specifications are stored internally in the HDToolBox program. If the drive you're using shows up on the list in the window, select it. Finally, click on OK to return to the main WBToolBox window. (The program will now put up a requester informing you that you're about to wipe out any data on the drive; with a new drive, this isn't a concern. Select the Continue button.)

If your drive doesn't appear in the internal list, you'll have to select the Define New Drive Type button. The Define/Edit Drive Type window contains text gadgets that let you enter the physical characteristics of your hard drive, which are detailed in the drive's documentation. If your drive is a SCSI drive you may yet be able to avoid entering information into all those text gadgets. Most SCSI drives store information about their configurations in their own ROM. By selecting Read Configuration From Drive, you're instructing HDToolBox to try and read the drive configuration information from the drive. Most SCSI drives support this feature. If your drive does, the system will read the information from the drive into the gadgets. Select OK to return to the Change Drive Type window, where you can select the drive from the list and hit OK to return to the main HDToolBox window.

If all else fails, you must enter the information by hand. The first text gadget is Filename and holds the name of the file that contains specifications for drives not directly supported by HDToolBox. Leave Filename on its default setting. The next three gadgets ask for the name of the drive manufacturer, the model name of the drive, and the revision number of the drive. It is not absolutely necessary that you enter all this information. You can get by with entering dummy information if you don't know the exact model name of the drive or the revision number.

With the next four gadgets, you have no leeway; you have to enter this information correctly for AmigaDOS to recognize and access your drive. If you can't find this information with the documentation that comes with the drive,

contact your dealer or the drive manufacturer directly. Cylinders refers to the number of storage tracks available on a single disk surface multiplied by the number of recording surfaces on all the disks in the drive. Many hard-disk drives contain more than one disk. Tracks are concentric storage areas on the surface of a disk. Heads refers to the number of read-write heads the drive contains. Drives often contain two, four, or more heads. Blocks per Track is the number of 512 byte blocks (sometimes called sectors) available on a single track, and Blocks per Cylinders is the number of blocks in a cylinder. This latter number is normally the blocks per track multiplied by the number of read-write heads contained in the system. Figure 6-9 shows a graphic representation of what these different values mean. If you've entered the information about the drive correctly, the size of the drive as it is listed in the window should be close to the size of the drive as listed by the manufacturer. In fact, most manufacturers understate the sizes of their drives by a small fraction.

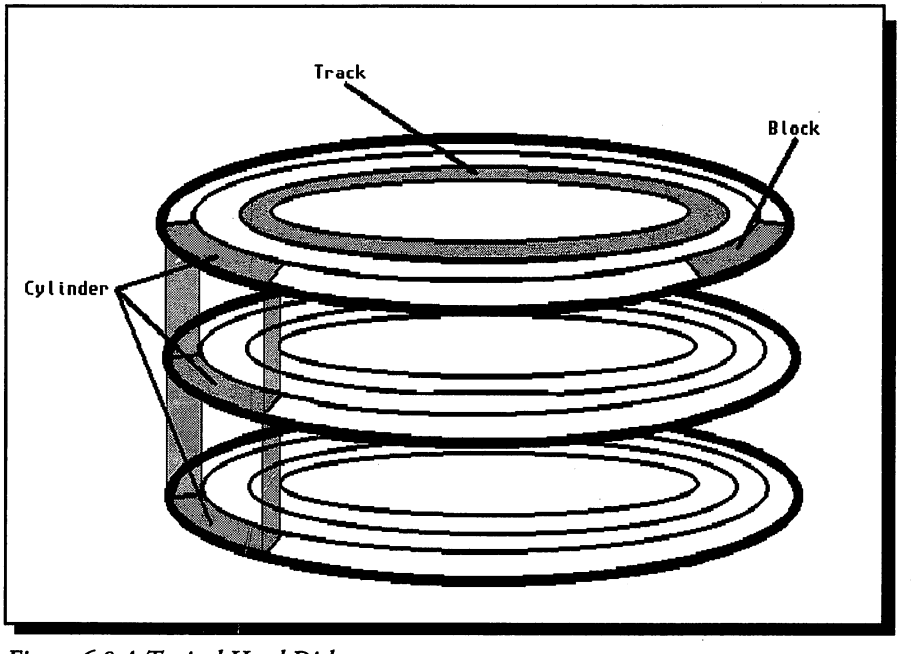


Figure 6-9 A Typical Hard Disk

A block — also called a sector — is the smallest physical partition on a disk. A track consists of a ring of blocks. A cylinder consists of corresponding blocks on the different recording surfaces in the hard-disk drive. A drive will have as many read-write heads as it has recording surfaces.

Of the other four gadgets in the window, you need concern yourself only with the one that asks if your drive supports reselection and the one that prompts you to enter the number of the cylinder where the read-write head parks when you power down your computer. The other two are not supported by the Amiga. The information on reselection and the parking cylinder is available in the documentation that accompanies the drive. If the drive automatically parks its heads, you needn't worry about entering the information manually. If you can't find the information on the park cylinder, use the number of the last cylinder on the system, which is the number of cylinders minus one.

After you enter the information about your drive, select the OK button or click Cancel if you want to exit without preserving the information. Both buttons return you to the Set Drive Type window; if you selected OK, the drive you just described will be listed in the scrolling list of drives. Select it from the list and hit OK. This brings up the requester informing you that you're about to lose any partitions on the drive. With a new drive, there are no partitions anyway, so select Continue. If you're working with an already partitioned drive, you should select Cancel and exit HDToolBox until you've backed up everything you need from the drive. The Set Drive Type window also lets you edit or delete any drive definitions you've previously made.

Bad Blocks

Because hard disk platters are expensive to make, manufacturers don't reject a disk if it has a few defects. Rather, the locations of the defects are noted either in written form or, in the case of most SCSI drives, in the ROM of the drive itself. A SCSI drive automatically reports this information to HDToolBox, which keeps AmigaDOS from using the blocks where the defects are located. If you have an ST506 drive or a simple SCSI drive, you may have to enter bad-block data by hand before you format the drive. Figure 6-10 shows the Bad Block Entry requester.

Low-Level Formats

When you first add a drive to your system, or if a drive is developing a lot of errors, you have to perform a low-level format. When you access this option, you receive a warning that a low-level format will destroy everything on a disk. You can choose to cancel the procedure at this point.

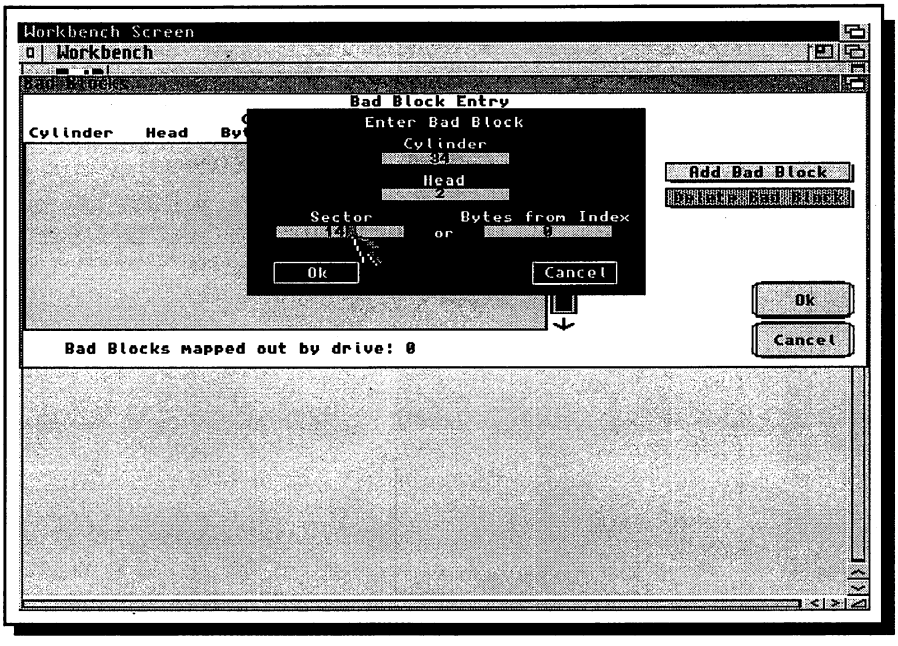


Figure 6-10 Bad Blocks

Although most SCSI drives map out bad blocks automatically, some drives require that you enter the information by hand. This is especially true of the ST506 controllers used in many Amiga 500s.

Partitioning a Drive

Selecting this option brings up the requester shown in Figure 6-11. This is a graphical representation of the partitions on you hard disk. The horizontal bar represents all the storage space on your hard disk, with a black bar representing the current partition, a gray bar representing unused space, and a hatched bar representing an already existing partition. You can set the size of the current partition by dragging the blue triangle, and change the current partition by clicking in another partition.

The Delete Partition button deletes the current partition, while New Partition lets you create a partition in unused space. Default Setup divides the disk into two equal-sized partitions. The Help button gives you some on-line help.

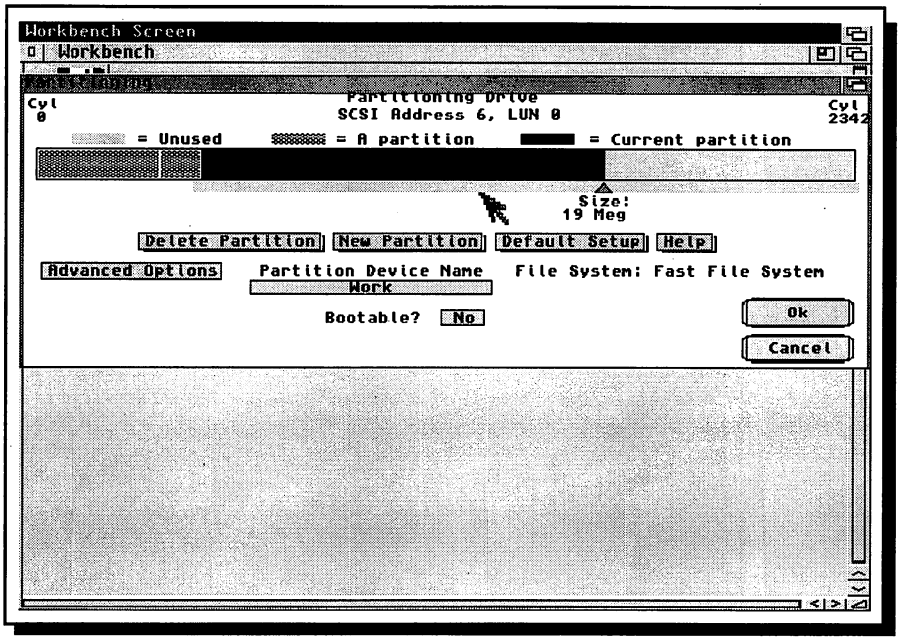


Figure 6-11 The Partition Display

HDToolBox provides a simple way to divide a hard disk into different volumes. This display shows that my hard drive consists of three partitions. I've made the largest one smaller before adding a fourth partition.

The Partition Device Name gadget lets you enter the AmigaDOS device name of the partition, and the Bootable? gadget lets you make it possible to boot from the partition. The File System label identifies the file system used by the partition. It should read Fast File System.

Selecting the Advanced Options button brings up the display shown in Figure 6-12. The Start, End, and Total Cyl gadgets let you precisely define the size of the partition in cylinders, while Buffers lets you enter the number of AmigaDOS buffers to give the drive (see AddBuffers in Chapter 9). If you have more than one bootable volume mounted on a system, the one with the higher Boot Priority becomes the volume that AmigaDOS boots from. Don't set a hard disk's boot priority above 4, because you will always want the option of booting from DF0:, which has a boot priority of 5.

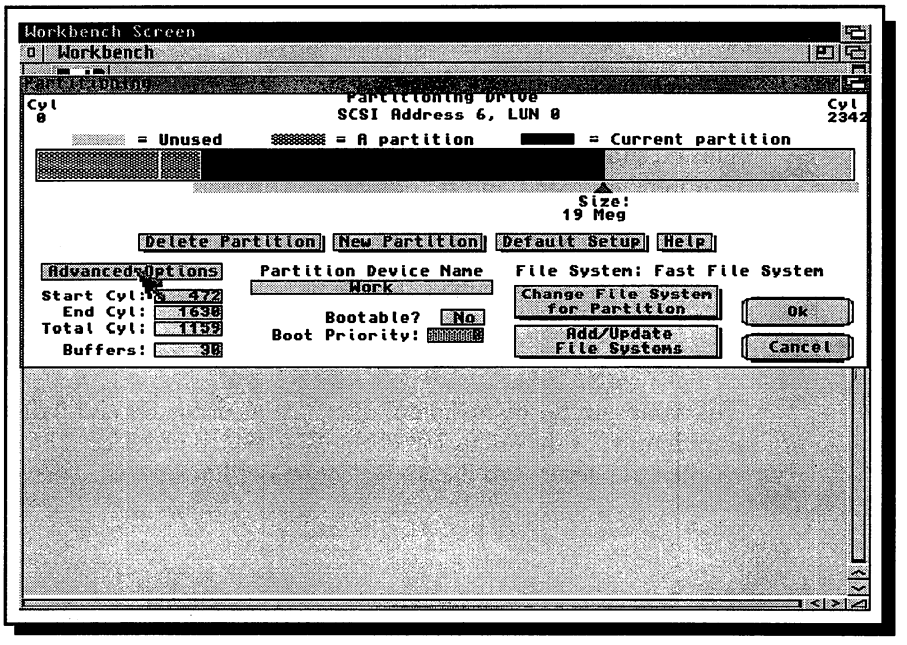


Figure 6-12 Advanced Options

If you want to create partitions of a precise size, you can enter the number of cylinders directly without using the graphic display. Advanced options also let you mount new file systems.

The Change File System button lets you designate a file system besides Fast File System. Unless Commodore comes out with an improved file system, you'll probably never use this option. The same goes for Add/Update File Systems, which lets you make other file system available to AmigaDOS.

Once you've set up your partitions, you select OK to return to the main HDToolBox window. To put your changes into effect, you select Save Changes to Drive. To cancel the changes, you simply select Exit.

Although Commodore's documentation discusses a Workbench-oriented hard-drive backup utility called HDBackup that is supposed to complement HDToolBox, the program didn't make it into the final release. Unless you want to try the Shell-only utility, BRU, described in Appendix C of the *Using the System Software* manual, you should check out one of the fine commercial backup utilities such as Quarterback from Central Coast Software.

Updating the Operating System

Because Amiga OS 2.0 was first released with a RAM-based Kickstart for the Amiga 3000, Commodore included a simple IconX project called Update2.X. This executes a simple script that lets you update to the latest Kickstart release. (For more on IconX and script files, see Chapter 10.) Once Kickstart 2.0 is committed to ROM, this project may no longer be supplied with Amiga OS 2.0.

Conclusion

That wraps up the discussion of the contents of the Extras2.0 disk and, more importantly, of the discussion of the Workbench interface. The next four chapters are devoted to the Amiga command-line interface, the Shell.

— 7 —

AmigaDOS and the Shell

When using the Workbench interface, you work with disk files by manipulating icons. You can execute a program file by double-clicking a tool or a project icon, or copy any file by dragging its icon from one window to another. Working in the Shell, however, is different. Unlike Workbench, which is a graphical-user interface (GUI), the Shell is a command-line interface, similar to those on MS-DOS and UNIX systems.

For a time, the Amiga was unique in offering a command-oriented *and* a graphical interface, but most major computers now offer both. For instance, Microsoft offers both MS-DOS and Windows for IBM PC and PC-compatible computers, while both UNIX International and the Open Software Foundation offer graphical interfaces for UNIX. Even Apple has recognized that no one interface is best for all applications: it plans to introduce a scripting capability with the next release of its operating-system software for the Macintosh.

What can a command-line interface give you that a graphical interface can't? First, greater flexibility. With a GUI you must specify each file or drawer you want to work with by selecting its icon. If you want to work on some files in a window but not others, you have to select the individual files. With a command line, you can use wildcard characters to specify files. For example, you can delete all files in a directory that relate to an already completed project with one command (provided you have used meaningful filenames). A second advantage of a command-line interface is the ability to automate tasks by stringing commands together in a script. Finally, there are some AmigaDOS functions that simply are not available from Workbench.

AmigaDOS Nomenclature

Just as Workbench is built upon the Amiga Intuition library, the Shell is built upon the AmigaDOS library, which controls access to disk drives, printers, and other devices attached to your Amiga. Thus, you will often hear people talk about "using AmigaDOS" (or, more simply, "using DOS"); what they are actually talking about is accessing AmigaDOS through the Shell.

With the Workbench interface, you manipulate objects called drawers, tools, and projects — all of which represent different types of files. The pictures on the icons relate to the function of the file, making it easier to choose the right icon for the right job.

With the Shell, you manipulate files by name. Because you don't have the visual clues to remind you what a file does, it is important that you give your files meaningful names. Because it is not concerned with pictures, the Shell also dispenses with the Workbench terms for different types of icons. Thus, a tool under Workbench becomes a program file when accessed from the Shell. Likewise, a project becomes a data file, a drawer becomes a directory file, and a volume becomes a disk or a partition on a hard disk. There is no Shell equivalent, however, to the Trashcan.

The most important attribute of a file — its name — can be made up of any combination up to 31 letters, numbers, and special characters, except for the slash (/) and colon (:). Some special characters, such as the question mark (?) and hash mark (#), are permitted, but using them will make wildcard pattern matching harder to perform, so I suggest you avoid them. When I have to separate words in a filename, I use a period (.) or an underscore (_). You can also include spaces in a filename, but this makes it harder to manipulate the file from the command line. I suggest that you also avoid spaces in filenames, even in those you access through Workbench. Examples of valid AmigaDOS filenames are:

90_budget
Startup-sequence
Letter.to.John.Doe
Chapter 2
Shell.info

Note that the fourth example above includes a space. Invalid filenames include:

Letter: John Doe
Chapter/2
THIS_NAME_HAS_MORE_THAN_THIRTY_ONE_CHARACTERS

Note that filenames can contain both uppercase and lowercase characters. AmigaDOS, however, does not distinguish between upper- and lowercase, therefore, the names Data_File, data_file, and DATA_FILE are all equivalent. Each filename in a directory, even if it is the name of another directory, must be unique. The system will keep you from creating two files with the same name in the same directory. If you move or copy a file into a directory that already contains a file with the same name, the new file will overwrite the older one. You can, however, have files with identical names on the same disk as long as they are not in the same directory.

Many operating systems make wide use of filename extensions. MS-DOS lets you use filenames that consist of eight characters, a period, and a three-character extension. The extension usually defines the file type. For example, LETTER.TXT indicates that letter is a text file, while BUDGET.DBF means that budget is a data file for dBASE III. Some Amiga programs also use file extensions, but they are not required for most AmigaDOS files — although icon files do need a .info extension in order to be recognized by Workbench.

The Shell Window

There are a number of ways to start up a Shell project from Workbench. You can open the Shell icon in the window of your system disk or open the CLI tool in the System drawer. You can even open a Shell by accessing the Execute Command item in the Workbench window. In this case, enter the command

```
NEWSHELL
```

into the text gadget and either press the Return key or select the OK gadget.

All of these methods bring up a Shell window and make it the active window (see Figure 7-1). Once you have a Shell window open, you can bring up others by entering the NewShell command on the command line. To follow the examples below, use a Shell window that you open by double-clicking on the Shell project in the window of your system disk.

Shell windows share many properties with Workbench windows. They contain a close gadget, zoom and depth gadgets, and a sizing gadget (although no scroll gadget), and all work the same as they do on other windows. Within its borders, however, a Shell window is very different from other windows. It is designed specifically to accept typed input from the keyboard and to display text output: Shell windows don't do graphics. In this regard, they imitate the text terminals used by most minicomputers and mainframes and the text-oriented command interpreters of MS-DOS and UNIX computers.

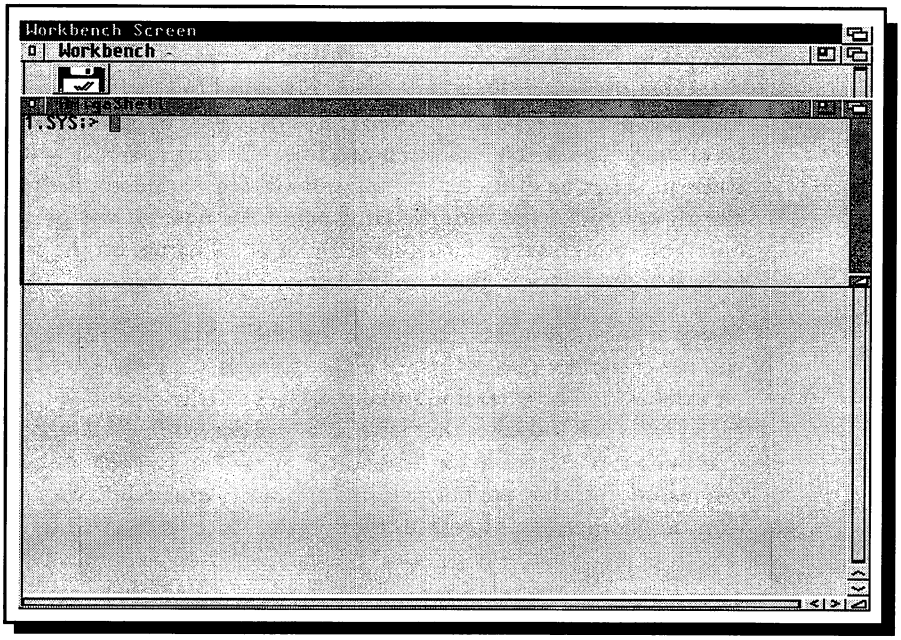


Figure 7-1 The Shell Window

The window you use to input commands to AmigaDOS, which has many of the same gadgets as Workbench windows. The biggest difference is the lack of scroll gadgets.

When you open a Shell window, the only thing you see is the prompt. The default prompt used with Amiga OS 2.0 lists the number of the Shell process and the name of the current directory followed by a prompt character (the greater-than sign [>]). Thus, when you open the Shell project on your system disk, your prompt looks like this:

```
1.Sys:>■
```

Following the prompt is a colored rectangle, the cursor.

In a Shell window, you can enter commands on only one line at a time — the line containing the cursor. The cursor marks the point on the line where the next character will appear after you type. It also marks the point where, if you wish, you can delete a character. As you enter commands and press the carriage return, the text in the window scrolls up a line. You cannot, however, scroll up to see text that has gone above the top of the window — although, as we will see later in this chapter, the Shell provides a way to recall previously executed commands.

Shutting down a Shell is a simple matter. Either select the close gadget at the upper-left corner of the Shell window, or enter

```
ENDSHELL
```

and then press Return.

Note that you cannot close down a Shell if you used it to run a program that itself has not shut down. While you can still type into the Shell window, pressing the Return key will have no effect except to move the cursor down one line. You can avoid this problem if you launch programs using the Run command.

Interpreting Input

To issue a command to AmigaDOS, simply enter the name of the command using your keyboard. For example, enter the following at the prompt:

```
DIR
```

and press the Return key. The Shell then executes the Dir (Directory) command and displays its output in the Shell window. Without any arguments that modify its behavior, Dir displays a list of all files — including directory files — contained in the current directory. Figure 7-2 shows the directory listing of the Sys: directory of an Amiga 3000. Compare this directory with Figure 7-3, which shows the window of the Amiga 3000 System2.0 disk. Note that any file with a .info extension shows up in the Workbench window as an icon, while files without .info extensions are not displayed. Files with the (dir) notation after their names are, in fact, directories themselves.

If the output of a command is scrolling by too fast in the Shell window, you can halt output temporarily by pressing the Spacebar. You can resume output by pressing the Backspace key. If you want to stop the operation of a command, press CTRL-C.

You may have noticed that just after you pressed the Return key above, the light for the disk drive holding your system disk came on. That was AmigaDOS loading the Dir command from disk. The commands you enter into the Shell are programs that exist on a disk, in memory, or in your Amiga's Kickstart ROM. When on disk, the AmigaDOS commands are kept in a special directory called the C: directory. (You will learn about this directory later.) You should note, however, that you are not restricted to running only AmigaDOS commands from the Shell. You can use the Shell to run just about any program you can launch from Workbench simply by entering the program's name on the command line.

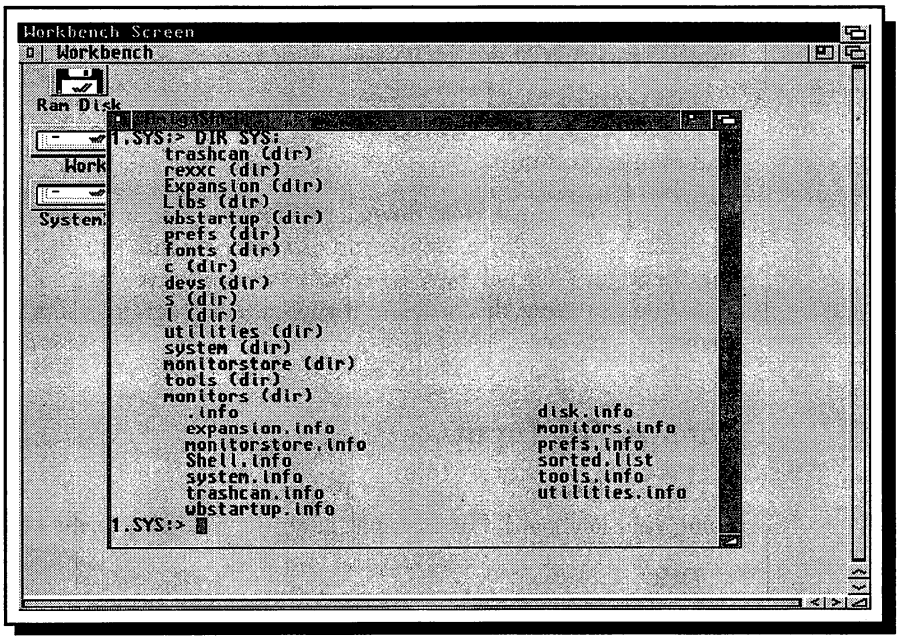


Figure 7-2 A Directory Listing

This is a listing of the root directory of the System2.0 disk. It shows an alphabetical list of directories followed by a two-column list of files.

As I mentioned before, AmigaDOS commands can accept arguments that alter what they do or how they do it. The Dir command is no exception. To demonstrate, enter the following:

DIR UTILITIES

and press Return. Here the output is different from that of the Dir command alone; it is a listing of the contents of the Utilities directory. You have modified the directory that is listed by the Dir command by passing it an argument — Utilities — that tells it what directory to list. If you do not pass Dir an argument, it defaults to listing the current directory.

In the example above, the command — Dir — and its argument — Utilities — are separated by a space. The space character has a special function in the Shell; it is used to separate the commands and arguments you enter. For example, assume you have used the New Drawer item from the Window menu to create a drawer called Word Processing Files. Now, in order to get a listing of this directory, suppose you entered

DIR Word Processing Files

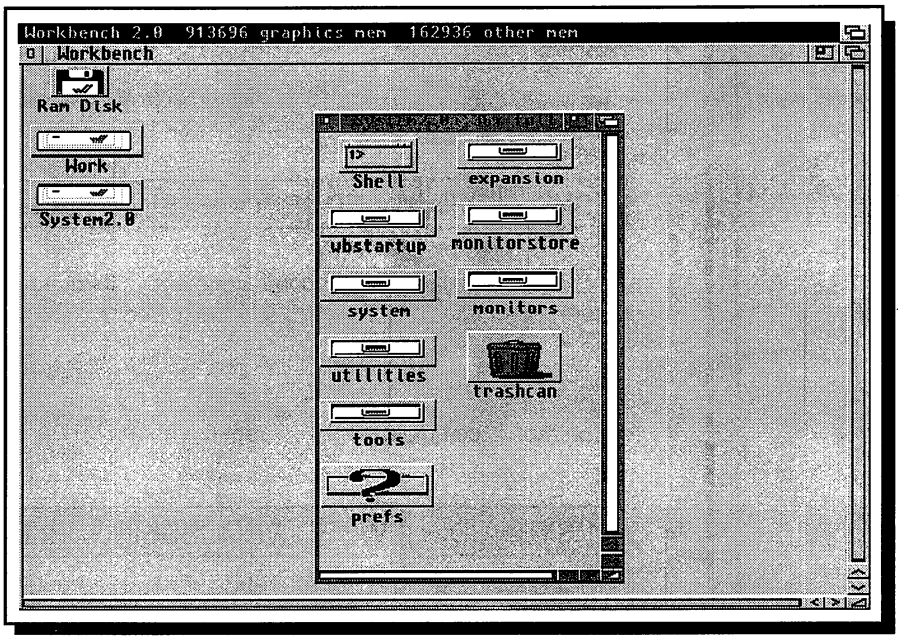


Figure 7-3 The System2.0 Drawer

Note the difference between the System2.0 listing (Figure 7-2) and its drawer. Only files and directories with .info files appear in the drawer window; all files and directories appear in a Dir listing.

followed by Return. You might expect to get a listing of the files in the directory Word Processing Files, but instead, you will get a message saying the system could not find the directory Word. You've just strayed from the procedure that the Shell uses to interpret what you input on the command line.

When you press the Return key after entering a command and its arguments, the Shell *parses* the characters (filters them from left to right) on the command line, looking for spaces. When it finds the first space in the line, it sends the characters before the space to AmigaDOS as the name of the command. (If there is no space in the command line, the Shell considers the entire line to be the name of the command.) The Shell then parses the line for other spaces and sends the characters between spaces, and between the last space and the end of the line, as the arguments of the command. (See Figure 7-4 for a graphic example of the parsing procedure.)

The problem, then, with the above example is that the Shell interprets the line

DIR Word Processing Files

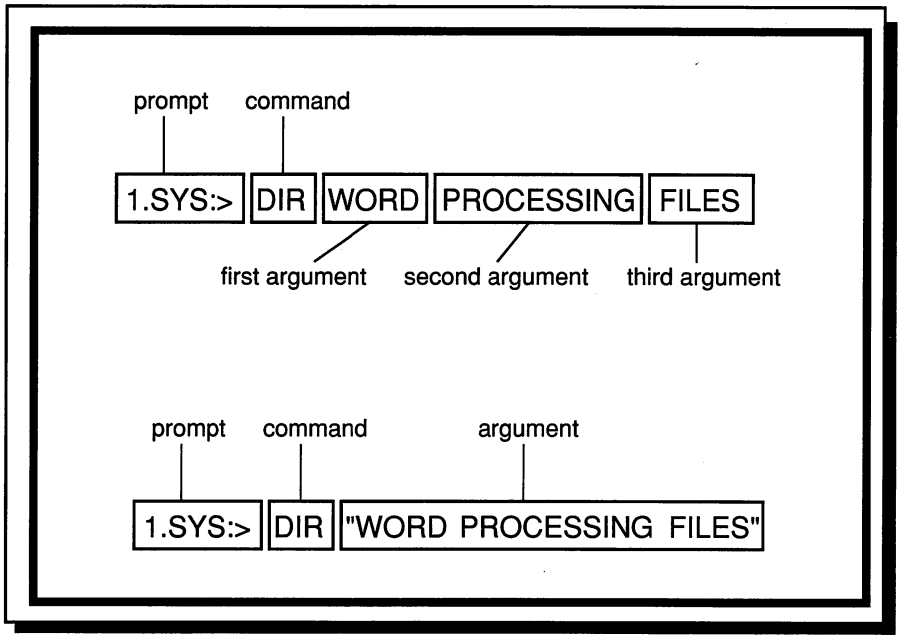


Figure 7-4 Command-Line Parsing

The Shell treats the space character as the delimiter between different words on the command line. From the beginning of the line to the first space is the command name; other words are arguments for the command. Note that by using quotation marks, you can get the Shell to treat names that contain spaces as a single AmigaDOS word.

as four separate groups of characters, or *words*. The first word, by default, is the name of the command. The next three are interpreted as the arguments you want to pass to the command. When Dir receives its arguments, it always takes the first one as the name of the directory you want to list. In this case, when it tries to list the directory named Word, which doesn't exist, it generates an error and produces an error message.

One solution — and the one I favor because I use the Shell frequently — is never to create a file, even using Workbench, that contains a space in its name. If you use Workbench primarily and like the freedom to include spaces in filenames, you can still access these files from the Shell by enclosing any filename containing spaces in quotes on the command line. The Shell will then interpret everything between the quotes as one word. Thus, the Shell interprets

DIR "Word Processing Files"

as two words: the command — Dir — and its one argument — Word Processing Files.

Note that the Shell does not begin to parse the command line until you press the Return key. To avoid repetition, from now on when I indicate that you should enter a command line, *I'll assume you understand you must follow up by pressing Return.*

Editing the Command Line

Unless you happen to be a flawless typist, occasionally you will make mistakes when entering commands and arguments into the Shell. One way to handle such errors is to press the Return key, let AmigaDOS give you an error message, and start again. Because some command lines are quite long, however, a better method is to edit the input before you press Return.

The Shell supports such basic editing keys as the Delete key, the Backspace key, the Cursor Left and Cursor Right (left-arrow and right-arrow) keys, and some control keys. Use the Cursor Left and Cursor Right keys to position the cursor on the command line. These are "nondestructive" keys, meaning they do not delete the characters they pass over on the line. Pressing the Shift-Cursor Left key combination moves you to the beginning of the line, while Shift-Cursor Right moves you to the end of the line.

The Delete and Backspace Keys *do* erase characters from the command line. DEL erases the character currently under the cursor, thus moving the characters to the right of the cursor one position to the left. Pressing the Backspace key moves the cursor one character to the left of its position, erasing the character in that position. Like the Delete key, it also moves the rest of the line one character to the left.

If you want to insert characters in the command line, you simply move to the appropriate spot and start typing. Each character you enter appears at the cursor position and moves the characters on the rest of the line one position to the left. The Shell does not support an overstrike mode, where entering a character replaces the character beneath the cursor. You have to use either DEL or the Backspace key to erase individual characters.

To erase more than one character at a time, and also to perform some special actions, the Shell lets you use a number of control-key (CTRL) functions. CTRL-W (pressing the Control key and w key simultaneously) allows you to delete the word to the left of the cursor; it deletes all the characters to the left of the cursor until it reaches a space character. CTRL-X and CTRL-B delete all characters on the current line. CTRL-K deletes the characters from the cursor to the end of the line, while CTRL-U deletes the characters from the cursor to the beginning of the line. Two control-key combinations perform the

same functions as dedicated keys: CTRL-H performs a backspace, and CTRL-M produces a carriage return.

One of the more interesting control-key combinations is CTRL-J, the linefeed, which allows you to enter two or more commands, one after the other, and have them execute sequentially. For example, to move from the Sys: directory to the Utilities directory and get a directory listing, enter:

```
UTILITIES<CTRL-J>
DIR
```

(I put the CTRL-J key combination in angle brackets because it is not a printable character.) When you enter the actual keystrokes, the cursor moves to the next line. Each line is treated as a separate command, but no line is sent to the Shell parser until you press Return.

To see how command-line editing works, consider the following example. Let's say you entered the following on the command line:

```
DOOR UTILITIES
```

Instead of pressing Return and having AmigaDOS tell you that it cannot find the command "door," simply press the Cursor Left key until the cursor is over the first o in door, press the Delete key twice, and then press i. Now, when you press Return, you will get a listing of the files and directories in the Utilities directory.

Note that once you have edited a command line, you do not have to move the cursor back to the end of the line before pressing Return. No matter where the cursor is, pressing Return sends the entire line to the Shell for parsing.

Copying Between Windows

In addition to editing and deleting characters on the command line, the Shell also lets you copy information between different Shell windows, and between the Shell and other character-based console windows, such as the windows created by the Ed editor. The first step in the copying process is selecting the text you want to copy. To select text, such as the output of a Dir command, select the beginning point of the text you want to copy with the left mouse button, drag the pointer to the end of the area you want to copy, and release the button. This action highlights the text between your beginning and end points (see Figure 7-5). You then press the right Amiga key and the x key together to copy the material to a temporary area. The original text is unaffected. (This is *copy* and paste, not cut and paste.)

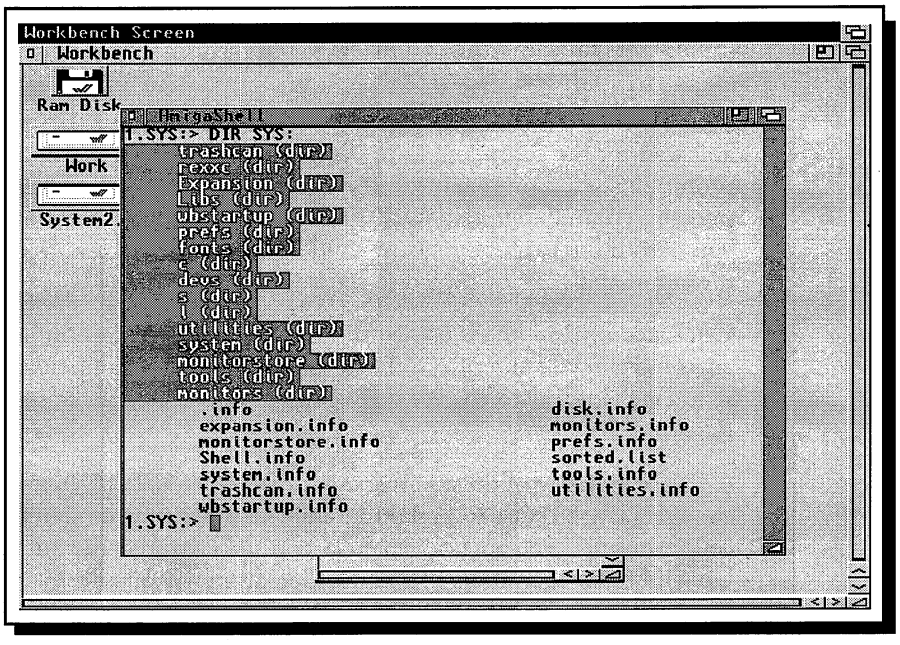


Figure 7-5 Highlighted Text

By dragging the mouse over the text in a Shell window, you select the text. You can then copy the highlighted text to any other AmigaDOS window or to the Ed editor.

You then move the pointer and activate the window you want to copy. With the cursor positioned where you want the incoming text to appear, press the Right Amiga and v keys together to copy the text in the window. Note that if you try to copy text on the command line of a Shell, the Shell will try to interpret each line of the copied material as a command line. This can lead to some bizarre error messages.

Command History

The Shell also supports the Cursor Up and Cursor Down (up-arrow and down-arrow) keys, but not as cursor-positioning keys. You use these keys to scroll through the command-history buffer.

Whenever you press Return on the command line, the Shell not only parses the line and sends the results to AmigaDOS, it also saves the line in a temporary storage area called the command-history buffer. This buffer contains the last 2,048 bytes you have input to that Shell. When you press the Cursor Up

key, you copy the last line you input from the buffer to the command line. For example, if you enter the command

```
DIR PREFS
```

and wait for the command to finish its output, you will get the command prompt back. If you then press the Cursor Up key once, the Shell will copy the previous command line — DIR PREFS — to the current command line. All you need do to execute the command again is press Return. When you do so, the command is again copied to the buffer.

Once a command line is copied from the buffer, you can edit it as you would anything else you enter from the keyboard. Thus, the command-history buffer is an excellent way to correct typing errors that you might miss before pressing Return.

As stated, the size of the command-history buffer is 2,048 bytes, which is enough space to store several dozen long commands with arguments, or a couple of hundred short commands. You scroll back in the buffer using Cursor Up, and forward in the buffer using Cursor Down. In effect, Cursor Up moves you backwards in time; Cursor Down returns you to your present position. When the buffer is full and you enter another command, the oldest line in the buffer is bumped to make room for the newest line.

The Cursor Up and Cursor Down keys also perform special functions when used with the Shift key. Shift-Cursor Down moves you to the bottom of the command buffer, while Shift-Cursor Up acts as a search function. The latter searches backwards through the command buffer for the most recent form of your current command line. The search can be for multiple words, single words, or just fragments. When a match is found for that line in the buffer, the entire line is copied to the command line.

AmigaDOS File Structure

In a previous example, you listed the contents of the Utilities directory by including the directory name as an argument on the command line. There are, however, other ways to list the contents of Utilities. The easiest is to simply make it the current directory.

I've used the analogy of an upside-down tree to describe the structure that the Amiga uses when it stores files on a disk, and the analogy holds true for the command-line interface. The current directory describes your position in the file structure. It is the directory on which the AmigaDOS commands operate

by default, unless your override that default by using command-line arguments.

When you first open a Shell using the Shell project or the CLI tool, you are at the top of the directory structure of your system disk — the root of the upside-down tree. In fact, AmigaDOS refers to the top-most directory of a disk as the root directory. (If you open a Shell by entering NewShell using the Execute-Command item, you will be at the root of the RAM: disk.) Unless you specify otherwise with a command-line argument, the commands you enter will work with and on this directory. You can change the current directory — thus changing the directory the commands work on by default — by entering the name of the directory to which you want to move. For example, if Sys: is your current directory — the one listed before the prompt — you can move to the Utilities directory by entering

```
UTILITIES
```

This changes the current directory to Utilities. Note that the prompt also changes to reflect the current directory. If you now enter

```
DIR
```

you get a listing of the Utilities directory.

In addition to entering pathnames, you can also use some special characters to move around in the directory structure. Entering a slash (/) moves you up one level in the directory structure; two slashes move you two levels up, and so on. Entering a colon (:) moves you directly to the root directory of the current disk. Thus, if you have moved to the Sys:Prefs/Env-Archive/Sys drawer and you want to return to the root directory, you can enter either

```
:
```

or

```
///
```

Note that when you return to the root of your system disk, the prompt displays the volume name of the disk (Workbench2.0 or System2.0) instead of its logical name, Sys:. Don't worry: The two names are equivalent, as you will see in the section below on directory assignments.

Because entering a pathname lets you move to any directory on any disk attached to your Amiga, it is a good idea to learn more about pathnames. To use this feature effectively, you need to have a good understanding of the different ways you can specify the location of a file or directory.

Filenames and Pathnames

Workbench is a direct-manipulation interface. To work on a file or to see the contents of a drawer, you must have the object in front of you. The Shell allows you to access files and directories that are not in the current directory and to move from the current directory to any other directory on any disk attached to your system — if you specify the proper pathname.

A pathname consists of a filename plus the directories and the disk above the filename in the AmigaDOS file structure. For example, your Prefs directory contains a program with the filename ScreenMode. In a floppy-disk system, its complete pathname is Workbench2.0:Prefs/ScreenMode. With a hard-disk system, its pathname is System2.0:Prefs/ScreenMode. (In both systems, the file probably has a second pathname, namely, Sys:Prefs/ScreenMode. That's because with both floppy- and hard-disk setups, the system disk gets the logical assignment Sys:. We'll have more on assignments later.)

Unlike filenames, each pathname on your system refers to one and only one file or directory. Because of logical assignments, however, one file can have many pathnames, but every pathname points to one file only. If you know the pathname of a file or directory, you can access it no matter where you are in the AmigaDOS file structure.

Let's dissect the pathname of the ScreenMode program. The first item in the pathname is the name of the disk where the file is located. Under AmigaDOS, the disk name is always followed by a colon (:). After the disk name come the names of the directory in which the file resides. Directory names are followed by a slash (/). Finally, the last item in the pathname is the filename. Figure 7-6 relates the pathname to the file's position in the AmigaDOS file structure.

If a file is nested within two or more directories under the root directory of a disk, you need to specify all the directories in its pathname. For example, the file that contains the bit pattern for the Times 24 font is named 24. It is located within the Times directory within the Fonts directory within the root directory of your system disk. Its pathname on a hard-disk system is therefore System2.0:Fonts/Times/24. Figure 7-7 relates this name to the file's location in the file structure.

If any component of a pathname — a disk, directory, or filename — contains a space, then the entire pathname must be enclosed in quotation marks when you enter it on the command line. For example, if you want to get a directory of the Env directory of the Ram Disk, enter

```
DIR "RAM DISK:ENV"
```

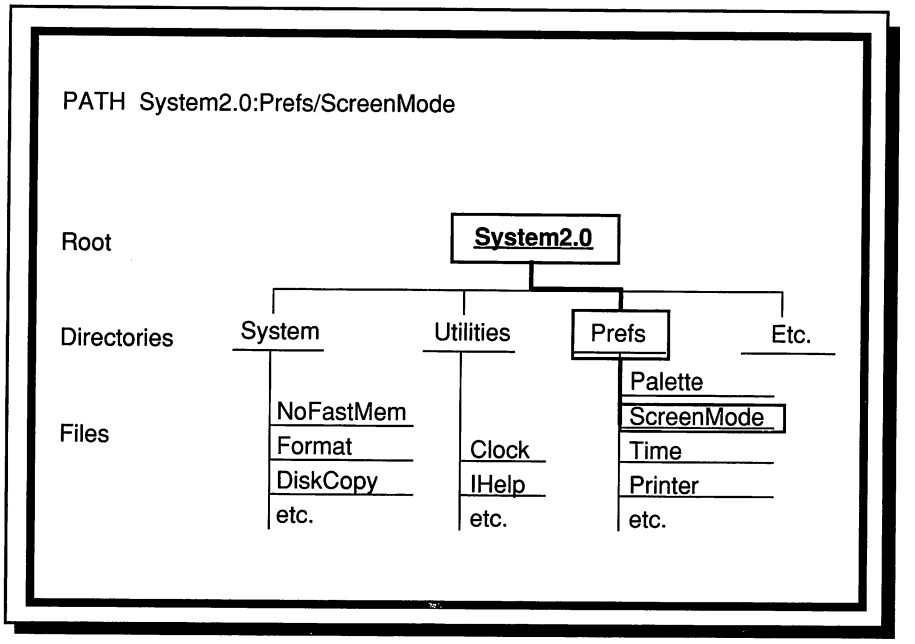


Figure 7-6 ScreenMode Path

The pathname for the ScreenMode program consists of its filename and all the directories above it in the file structure. In this case, the pathname consists of the volume name, one directory name, and the filename.

Note that I did not put a slash after the name of the Env directory. When a directory name is the last item in a pathname, you do not have to include the slash after its name. The same does not hold true for disk volume names. These must *always* be followed by a colon.

Relative and Absolute Pathnames

Pathnames come in two flavors: relative and absolute. Absolute pathnames are those that include the name of the disk where the file resides. No matter what the current directory, you can access any file on the system if you know its absolute pathname.

A relative pathname is a different matter. It does not describe the absolute position of the file in the file structure, but only its position relative to the current directory. Thus, if your current directory is the Fonts directory of your system disk, you can get a listing of the files in the Times directory in two ways — by entering the absolute address:

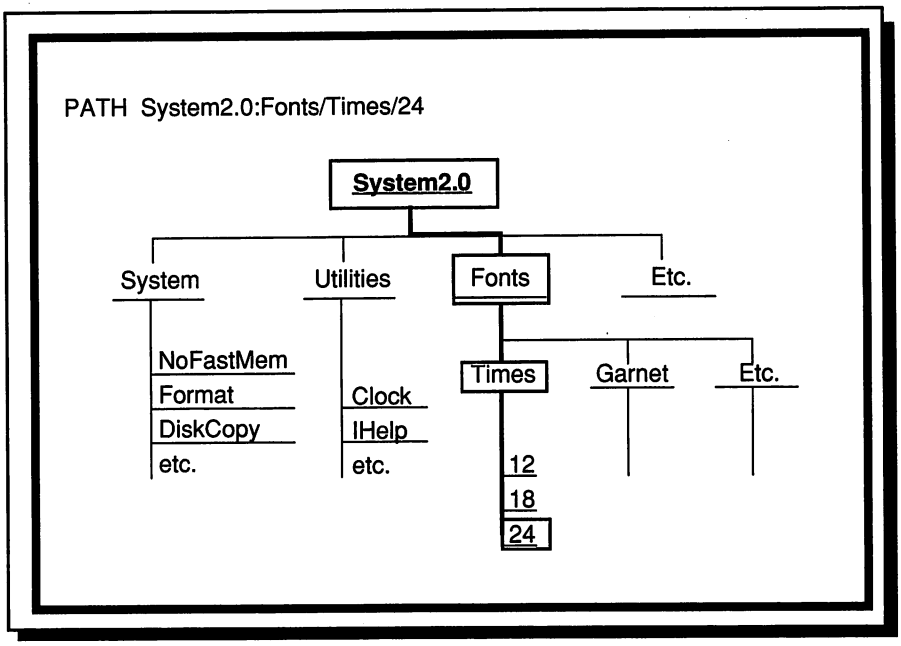


Figure 7-7 Finding the Times 24 Font

The Times 24 font is in a file named 24 in the Times directory, which is in the Fonts directory in the system disk. The file 24 is one level deeper in the disk file structure than is ScreenMode.

```
DIR System2.0:FONTS/TIMES
```

or by entering the relative address:

```
DIR TIMES
```

Relative pathnames work only if the file or directory you want to access is at or below the current directory in the file structure (see Figure 7-8) In effect, the system combines the relative pathname with the path of the current directory to produce an absolute pathname.

Directory Assignments

In addition to relative and absolute pathnames, AmigaDOS gives you another method for accessing particular directories, namely, using a shortcut called a logical assignment.

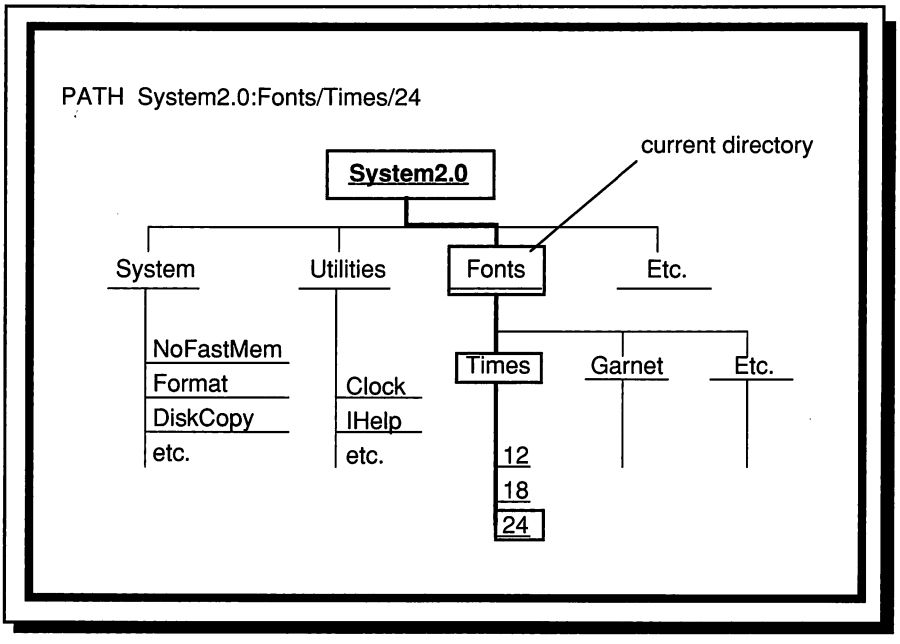


Figure 7-8 Relative Pathname

By moving to the Fonts directory on the system disk, you can access the 24 file without entering the absolute pathname. You can instead use that part of the pathname relative to the current directory.

A logical assignment is a name that replaces a pathname. For example, if you have a directory with the pathname System2.0:Word.Processing/Data/Letters/Mom, you can, using the Assign command, equate this pathname to a logical name such as Mom: by entering the following:

```
ASSIGN MOM: System2.0:Word.Processing/Data/Letters/Mom
```

From then on, whenever you use Mom: in the Shell, AmigaDOS automatically substitutes the real pathname for the logical one. If you have directories that you access often, you might consider giving them logical assignments.

Logical assignments are more than just a way to cut down on your typing. AmigaDOS makes 10 logical assignments when you boot your system. These assignments point to the location of various system resources that any person or program can access. The default assignments contained within the Kickstart ROM are:

SYS:	The disk you booted from, either Workbench2.0 or System2.0
C:	Sys:C
S:	Sys:S
FONTS:	Sys:Fonts
DEVS:	Sys:Devs
LIBS:	Sys:Libs
L:	Sys:L
ENVARC:	Sys:Prefs/ENV-Archive

In addition, the standard Startup-sequence makes the following assignments:

ENV:	RAM:Env
T:	RAM:T
CLIPS:	RAM:Clipboards

Using logical directory names, any user or program can access basic system resources. For example, if a program wants to load a font, it does not have to know the absolute pathname of the Fonts directory; it need only use Fonts: directory.

The defined purpose of each of the default assignments is as follows:

SYS: The Sys: directory is assigned to the root directory of your boot disk. It provides a simple shorthand for accessing what is probably the most important disk on your system, and it is used by the system to make the other default assignments. It is also the current directory of all Shells you open with the Shell or CLI icons.

C: The C: (Commands) directory is the default location of the AmigaDOS disk-based commands. When you enter a command into the Shell, AmigaDOS first checks to see if the command is one of its internal commands or if the command has been made resident. It then checks the current directory. If it cannot find the command in these places, it checks for it in the C: directory. If you have a program that you want to be able to run easily no matter what directory you are in, you can simply put it into the C: directory. You can add to the places where AmigaDOS searches for commands — its so-called search path — using the Path command.

If you have a one-drive system and want to use AmigaDOS to work with a disk besides your system disk, you may find yourself being prompted to perform a lot of disk swapping to enable AmigaDOS to load commands from the C: directory. Be patient: Further on we will learn how to avoid a lot of this kind of disk swapping.

The internal AmigaDOS commands and the contents of the C: directory are the primary focus of this section of the book. Appendix A provides a quick reference to these commands.

S: The S: (Scripts) directory is the default location of AmigaDOS and other script files. A script is a series of commands that you can execute by accessing the script file. The most important script file in the S: directory is the Startup-sequence file, which AmigaDOS executes every time you boot your computer. Other scripts include ED-startup and Shell-startup. You'll learn more about scripts in Chapter 10.

FONTS: This directory is the default location for font files and directories. The font files contain the sizes of the fonts available in a particular typeface. For example, Courier.font lists the pixel sizes of the available Courier fonts. The Courier directory actually holds the font files, which have numbers for names. The numbers correspond to the vertical size of the font in pixels.

LIBS: This directory stores disk-based libraries. A library is a collection of routines that perform related functions. For example, the Intuition library consists of routines that create and manage the Amiga windowing interface. Most Amiga libraries reside in ROM; some, less-frequently used libraries reside in the Libs directory and are loaded by programs that need them. Some third-party software programs even come with their own libraries that you must move into the Libs: directory.

L: The L: directory is used as the default storage place for device handlers. Handlers are software routines that interface the operating system to certain hardware features or redefine how the system works with hardware. For example, the RAM_handler lets AmigaDOS access main memory as if it were a disk drive.

DEVS: The Devs: directory contains device drivers, which allow AmigaDOS to interface with peripheral devices using a standard set of commands. Inside the Devs: directory, for example, are the keymaps and printers directories, which store the definitions for the alternate keymaps you can use on your system and the printer drivers available with the system, respectively. (See Chapter 9 for more on AmigaDOS devices.)

ENVARC: This directory contains the default location for the settings files you create with the Preferences editors. The files themselves are stored in the Envarc:Sys directory. In addition, any icon images that you wish to use in place of the default images supplied by Workbench for files without .info files are stored here.

ENV: The Env: directory is used to store environment variables. In addition, AmigaDOS copies the Envarc:Sys drawer to Env: during startup. ENV:Sys

always contains the Preferences settings and default icon images currently in use.

CLIPS: This directory is used by programs that support the Amiga clipboard device, which allows you to cut and paste text and graphics between the windows of applications. When you cut or copy text from a window, it winds up in Clips:, where any other program will know where to find the material.

T: T: is a directory used to store temporary files. Many programs create and discard temporary files during execution. Assigning T: to the Ram Disk ensures that the temporary storage location is always available.

Until you have a lot more experience with AmigaDOS, I recommend that you do not change the default assignments or delete or rename any of the directories used in those assignments. Doing so can produce unusual results. At any time, you can see the logical assignments currently active on your system by entering:

```
ASSIGN
```

The output of the Assign command (Figure 7-9) lists the volumes mounted on your system, the currently active directory assignments, and the devices mounted on your system.

Disk Names

Under Workbench, you access a disk by clicking on an icon with the disk's volume name. The volume name refers to the disk itself, not the drive it was in. Under AmigaDOS, you can also access a disk using its volume name, although you will have to use quotation marks if the name, such as Ram Disk, contains a space.

In addition to volume names, AmigaDOS also lets you access a disk by using the name of the disk drive that holds the disk. This name is called the device name. For example, if you have a single-floppy system, your disk drive has the device name DF0:, for disk floppy 0, and your system disk is named Workbench2.0. With Workbench2.0 in DF0:, you can refer to the disk by using either its volume name or the name of its device. If you change disks, DF0: will refer to the new disk in the drive, while Workbench2.0 will still refer to that particular volume. Note that you can have two disks with the same volume name active on your system. AmigaDOS can tell disks apart even if they have the same name.

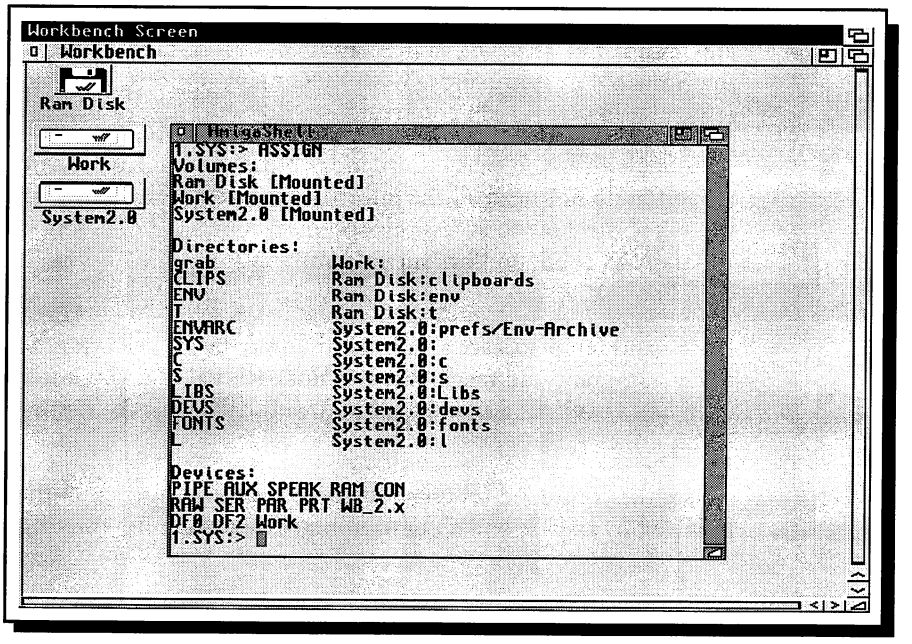


Figure 7-9 Logical Assignments

The Assign command lists anything on your system that is followed by a colon. This includes volumes, assigned directories, and devices.

If you add an external floppy drive to an Amiga 500 or a second internal drive to an A2000 or an A3000, this drive is called DF1:. A second external drive on an A500 or the first external drive on an A2000 or an A3000 is called DF2:.

The Ram Disk whose icon appears when you boot your system also has a device name — RAM:. You will see this designation used most often when working in AmigaDOS because it does not contain a space. Hard drives and partitions also have device names, and you can use these in place of the volume names if you wish. The standard Amiga 3000 hard drive comes with two Amiga OS 2.0 partitions: System2.0 and Work. The device names of these volumes are WB_2.x: and WORK:

Command Templates

Between the internal commands now stored in ROM (all commands were disk-based in earlier versions of AmigaDOS) and those in the C: directory, you

have access to over 70 AmigaDOS commands. Most have numerous options and arguments, resulting in literally thousands of different ways you can use the AmigaDOS commands. With these many options, your main problem is remembering what each command can do.

This book gives you that information, but once you have learned a command, you often do not need all the information given here. You simply need a reminder about how to access the options a command offers. Just about every AmigaDOS command has such a reminder built in — in the form of a command template.

Command templates are a shorthand notation that tells you the options that a command can recognize. It is not essential that you know how to decipher a command template, but it can make your computing a lot easier if you do.

To see the command template for a command, simply enter the command, followed by a space and a question mark. For example, entering

```
DIR ?
```

brings up the following:

```
DIR,OPT/K,ALL/S,DIRS/S,FILES/S,INTER/S:
```

The cursor sits just after the template, waiting for you to enter an argument for the command. I'll use the template of the Dir command as an example of how to make sense of a command template because it has both standard and non-standard features.

When used alone, the Dir command brings up a sorted list of the directories and files contained in a directory. Directories are listed first, with a (dir) marker, followed by the files in two columns. The template lists the ways you can modify the operation of the command.

The various arguments you can send to the Dir command are separated by commas in the template, but do not use commas when you enter them. Instead, you should separate the arguments with a space. Arguments are represented by a keyword, which may or may not be required on the command line in order to access the argument. Many keywords are followed by modifiers — such as /A or /S — that give more information about the keyword or the argument itself.

The first argument for the Dir command is defined by the keyword DIR, which indicates that you can enter the pathname of any directory that you wish to list. Thus, if you enter

```
DIR DIR RAM:
```

you will get a directory of the root directory of your Ram Disk. In this example, the first DIR is the command name, the second DIR is the argument keyword, and RAM: is the argument itself. Because the keyword DIR is not followed by a /K modifier, you do not have to include it with the argument. You can get the same results by entering the following:

```
DIR RAM:
```

This is the usual form of the Dir command. Except when testing the command for this book, I've never used the DIR keyword with the DIR argument.

Note that the DIR keyword also lacks a /A modifier. This means that the argument itself is optional. As you know, when you enter the Dir command without an argument, it gives you a directory listing of the current directory.

The next argument is OPT/K. The /K modifier means that if you use the argument, you must include the keyword on the command line. Using the argument is optional, however, because the keyword is not followed by a /A modifier. OPT stands for options; it lets you access some of the predefined options of the Dir command.

These options are available by entering the proper letters on the command line. They are A, I, AI, D, and F. (Note that these letters are not listed in the template, which is why the world needs reference books.) A brief explanation of these options is as follows:

A: This option stands for all. When you use it, as in

```
DIR OPT A
```

it causes Dir to list every file in every directory below the current directory, indenting each list of files to reflect its relative position in the file structure (see Figure 7-10). If you enter this command from the root directory of a disk, you will get a list of every file on the disk. Note that you can have multiple arguments on a command line. Thus, to see all the file in and below the Fonts: directory, enter

```
DIR FONTS: OPT A
```

I: The Dir command uses the I option to produce an interactive directory. It will list each file and directory in a directory, one at a time, followed by a question mark. At this point, you have seven options:

Return: Pressing the Return key moves you to the next item in the interactive directory listing.

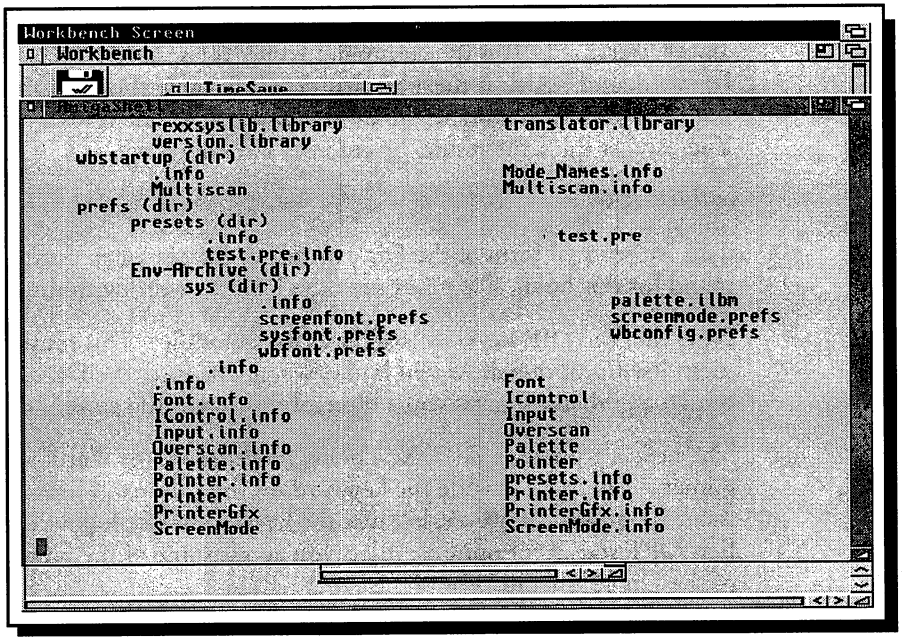


Figure 7-10 The ALL Option

You can get an idea of the AmigaDOS file structure by using the ALL option with Dir. It indents directory and filenames to reflect how deep they are in the file structure.

E: Pressing this key when a directory name is displayed will move you into that directory, where the interactive listing continues with the first item in the directory. The listing will go through every item in this directory before returning to the upper directory to continue the list.

B: This key returns you to the upper directory immediately, without listing the rest of the contents of the present directory.

Q: Quits interactive mode.

DEL: The three letters, not the Delete key. Entering them deletes the current file.

T: This key displays the current file on screen. You can press CTRL-C to abort the display and return to the directory listing.

?: Entered at the question-mark prompt, this character lists commands available in interactive mode.

COM: Entering these letters, or simply C, lets you suspend the interactive directory and execute most AmigaDOS commands. You will be prompted to enter a command on the next line, or you can enter COM or C followed by the command in quotation marks on the same line. Commodore cautions that

you should not start another interactive directory using the C option, and that you should avoid formatting a disk with it, because when called in this manner, Format does not put up any confirmation requesters. When the command you indicate finishes executing, you return to the interactive directory.

AI: This option produces an interactive directory identical to the I option. The only difference is that when you use the Q option to quit interactive mode, you get the rest of the directory in All mode.

D: The D option lists only the directories contained in the indicated directory.

F: This option lists only the files in that directory.

The other four arguments in the Dir command template access the same functions as the corresponding OPT options. All of them — ALL (OPT A), DIRS (OPT D), INTER (OPT I), and FILES (OPT F) — are followed by the /S modifier. This identifies the keyword as a switch, meaning that the option is only active if you include the keyword on the command line. Thus, if you enter:

```
DIR ALL
```

you get the same listing as you get with:

```
DIR OPT A
```

The Dir template does not tell you everything about the command — it does not list the OPT letters or the interactive commands — but it does give enough information about the command to serve as a gentle reminder when you forget how to access one of its arguments. You simply need to remember that the OPT options are identical to the first letters of the switches, and that the interactive commands are available at the question-mark prompt.

Keyword Modifiers

The arguments of the Dir command do not use all the keyword modifiers used by AmigaDOS. The complete list of modifiers is as follows:

/A: Indicates that the argument described by the keyword is mandatory. The command cannot execute without the argument.

/K: Indicates the keyword is mandatory if you want to use the argument. Otherwise, the keyword is optional.

/S: The keyword is a switch. To use it, you must include the keyword on the command line, and you must include it every time you want to access that option.

/N: Indicates that the argument must be a numeric value.

/M: Indicates that you can enter as many of the arguments of the indicated type as you like.

/F: Indicates that the argument must be the final one on the command line. This modifier is used only with arguments that are character strings. With this modifier, the string may contain spaces; you do not have to use quotation marks.

Many arguments have more than one modifier. For example, a keyword followed by /A/K means that both the argument and its keyword must be present. /K/N indicates that the keyword must always be present, and that the argument is a numeric value.

Other characters you will see in templates include a comma (,), which means that the command has no template, and an equals sign (=), which when used in an argument indicates two equivalent keywords for the same argument.

AmigaDOS commands expect their arguments in the order they appear in the template. With the Dir command, for example, the command expects the DIR argument before any of the options. You can alter the order of the parameters by using the keywords with the arguments, even when the keywords are not required. Thus,

```
DIR Prefs FILES
```

and

```
DIR FILES DIR Prefs
```

yield the same results.

The AmigaDOS commands use many different keywords in their templates. Listing them in one place is not useful because the functions described by keywords are often specific to a certain command. I define the keywords a command accepts when I introduce the command and also in Appendix A.

Pattern Matching

Many AmigaDOS commands let you perform some action on more than one file or directory at a time through the magic of pattern matching. Here you may substitute one or more characters in an argument with wildcard characters that can stand in for any character. For example, if your current directory has

three IFF picture files named mountain.pic, ocean.pic, and still-life.pic, you could copy all three files to the Ram disk with a single command:

```
COPY #?.pic to RAM:
```

#? defines a pattern. The Copy command acts upon any file that matches the pattern.

The special characters used in pattern matching are the question mark, pound sign, vertical bar, parentheses, and apostrophe. A pattern can be any combination of normal characters and special ones. An explanation of these special characters is as follows:

?: The question mark stands in for any one character. For example, if you entered

```
COPY file.? TO SYS:
```

when the current directory contained file.1, file.12, file.2, oldfile.1, and file3, the files copied would be file.1 and file.2. The other filenames do not match the pattern.

#: The pound sign matches zero or more occurrences of the character that follows it. Thus, the pattern #file matches all the following:

```
ile
file
ffile
fffile
```

The pound sign and the question mark combine to form the ultimate in pattern matching. When used together they stand in for any number of any characters. Thus,

```
COPY Utilities/#? TO RAM:
```

copies every file in the Utilities drawer to your Ram Disk. Together, these two characters are the equivalent of the wildcard character (*) used by MS-DOS and other operating systems.

|: When two patterns on the command line are separated by a vertical bar, AmigaDOS will act on any file that matches either of the patterns. In the following example,

```
COPY Utilities/#?c#?|#?er TO RAM:
```

copies any file that has a c in its name and any file ending with er.

(): You use parentheses to group patterns together to form a larger pattern. For example,

```
COPY let#?(txt|doc) to RAM:
```

matches all files whose names begin with let, are followed by any number of any characters, and end with either txt or doc.

%: The percent sign matches a null string. It is used most often with the vertical bar when you are not sure that a filename will have a certain number of characters. For example,

```
COPY (a|o|%)pict#? to RAM:
```

copies to RAM: any files whose names begin with apict, opict, or pict.

': The apostrophe is used to call off pattern matching. It indicates that the character that follows should be taken literally and not as a pattern. For example, if you wish to list the contents of a directory named Budgets?, you can enter

```
DIR Budgets'?
```

The apostrophe indicates that the question mark is part of the filename and not a pattern for matching.

~: The tilde negates the following pattern. For example,

```
DIR~#?.info
```

lists all the files that *do not* have .info extensions.

Pattern matching using wildcard characters is a very powerful feature. It lets you use a single command, such as Copy, Type, or Delete, to act on many files at once.

Console Window

If you are familiar with pattern matching on other computers, you know that the asterisk (*) is the most common wildcard character on other systems. On the Amiga, the asterisk has a different meaning; it refers to the current Shell window.

By using the asterisk, you can tell a command to get its input or display its output in the current window, instead of where it normally does. For example, the Copy command usually takes one file and copies it to another location. You can change that procedure by entering the following:

```
COPY * TO My_File
```

After entering this line, whatever you type into the Shell is copied to a file named My_File. To stop the operation and return to the Shell prompt, you have to enter the AmigaDOS end-of-file character, CTRL-\.

You can also reverse the process and copy a file to the Shell window with

```
COPY My_file TO *
```

In effect, this lets you use the Copy command as a Type command.

Command Redirection

Because the Shell uses a console device, all AmigaDOS commands expect to get their input from the keyboard and to display any output in the Shell window. These are the standard input and standard output devices for the Shell. You can direct that a command get its input from another source — or send its output to another destination — by using command redirection.

The command redirection characters are the greater-than (>) character and the less-than character (<), also called the right- and left-angle brackets. The greater-than character redirects a command's output, while the less-than character redirects its input.

If you use a redirection character on a command line, it should come after the command and before any arguments. It must be separated from the command by a space, although it need not be separated from the first argument on the command line.

Command redirection has many uses. One of the more common is in conjunction with the Type command, as in

```
TYPE >PRT: My_File
```

As you will see in Chapter 8, the Type command normally outputs the contents of a file to the screen. In this case, however, you have redirected the output of the command to PRT:, which is the AmigaDOS name for the Amiga printer device. Thus, this command outputs the contents of My_File to your printer.

Note that command redirection affects the current command line only. The next line you enter will use the standard input and output devices, unless you again use a redirection character.

Another common use of output redirection is to send the output of a command to a file. For example,

```
DIR >disk.catalog SYS: ALL
```

sends the output of the Dir command to a file named disk.catalog. If the file does not exist, it will automatically be created. The file will contain the names of every file and directory on your system disk.

The most common use of output redirection is in conjunction with the NIL: device. The NIL: device is a "bit bucket," a place where you send unwanted output. Many commands produce messages after performing their functions. The Copy command tells you about every file it copies, while the Delete command informs you of all the files it has deleted. Frequently, especially when executing a script, you do not want to see these messages. To get rid of them, simply redirect the output of the commands to the NIL: device. Your startup-sequence has at least three examples of redirecting output to the NIL: device. One of these

```
ENDCLI >NIL:
```

is the last item in the Startup-sequence file. It suppresses the informational message that you normally get when you close a Shell. (Note that EndCLI and EndShell are equivalent.)

Input redirection is much less common than output redirection, simply because most commands take so little input that it makes no sense to get the input from a file or another source.

One final form of redirection involves the use of the append operator (>>). By using it to redirect output to a file, you do not overwrite the previous contents of the file; instead, you append the new material to the end of it. For example, create a file named command.list by entering

```
DIR >command.list C:
```

When you output the file using the Type command, you will see a list of the files in C:. If you then enter

```
RESIDENT >command.list
```

the output of the Resident command will overwrite the previous contents of the file. To prevent this, use the append operator:

```
RESIDENT >>command.list
```

This adds the output of the Resident command to the previous output of the Dir command. The resulting file lists all 73 of the AmigaDOS commands.

Conclusion

With the information in this chapter, you should be able to use the Shell interface effectively and efficiently. The following three chapters tell you how to use the AmigaDOS commands. Chapter 8 describes the commands that provide information about your system and let you change the system configuration. Chapter 9 covers the commands that let you manipulate AmigaDOS files and devices. Chapter 10 finishes the discussion of AmigaDOS with a look at command scripts.

— 8 —

Delving into AmigaDOS

You find AmigaDOS commands in two places. The primary location is the C: directory, which, unless you changed it, is assigned to the Sys:C directory. On the Amiga 3000 release disk, this directory holds 43 AmigaDOS commands; the number probably won't vary in the releases for the Amiga 500 and 2000. You can view these commands by entering:

```
DIR C:
```

Figure 8-1 shows the output of this command.

In addition, with Amiga OS 2.0, 30 AmigaDOS commands are stored in the system's Kickstart ROM and are thus always available. You can see these commands by entering:

```
RESIDENT
```

The commands marked Internal (see Figure 8-2) are those contained in Kickstart. Notice that three commands found in the C: directory — Assign, List, and Execute — also show up in the list of resident commands, although they are not labeled as being internal. These are actually disk-based commands that are made resident by the Workbench2.0 and System 2.0 Startup-sequence. You'll learn more about the Resident command later in this chapter.

For purposes of organization, I've divided the 73 AmigaDOS commands into three categories and devoted one chapter to each category. This chapter deals with commands that give you information about the current state of AmigaDOS and, in many cases, let you change that state. (I call these the information and configuration commands.) Chapter 9 deals with commands that let you modify files and devices, while Chapter 10 describes how you create and modify command scripts.

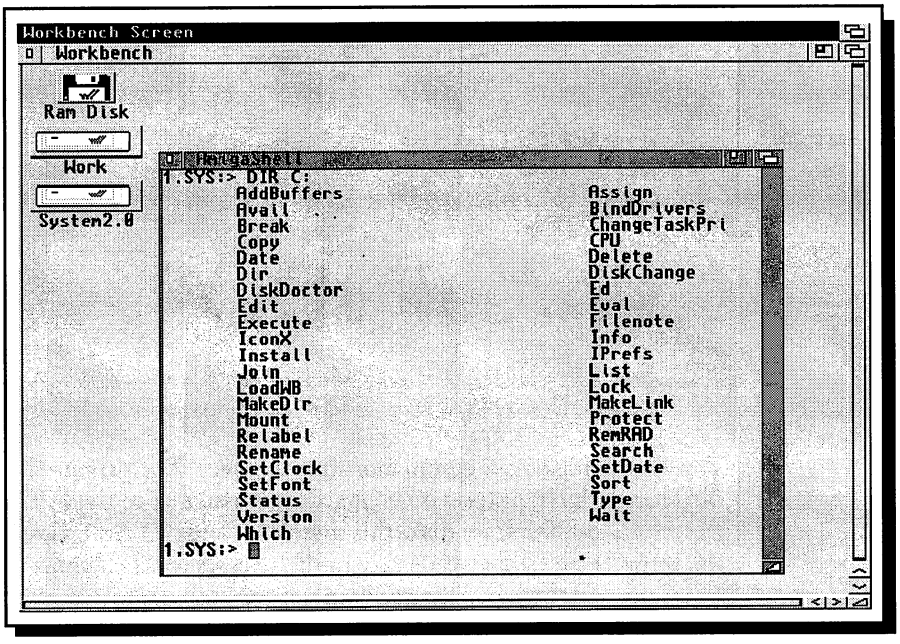


Figure 8-1 Disk-Based Commands

The C: directory contains the disk-based AmigaDOS commands. Whenever you use such a command from the Shell, AmigaDOS must load it from disk before executing it.

File Information Commands

From the last chapter, you already know about Dir, the most commonly used information command. It returns information about the files and directories on the disks mounted on your computer. Other commands that can return information about any file or directory are List, Assign, Resident, CD, Type, and Search.

List: Listing Files

The List command is a more sophisticated version of Dir. In addition to listing the files in a directory, it displays other useful information. Dir is best for finding a particular file; List gives you much more information. Figure 8-3 shows some example output of a List command.

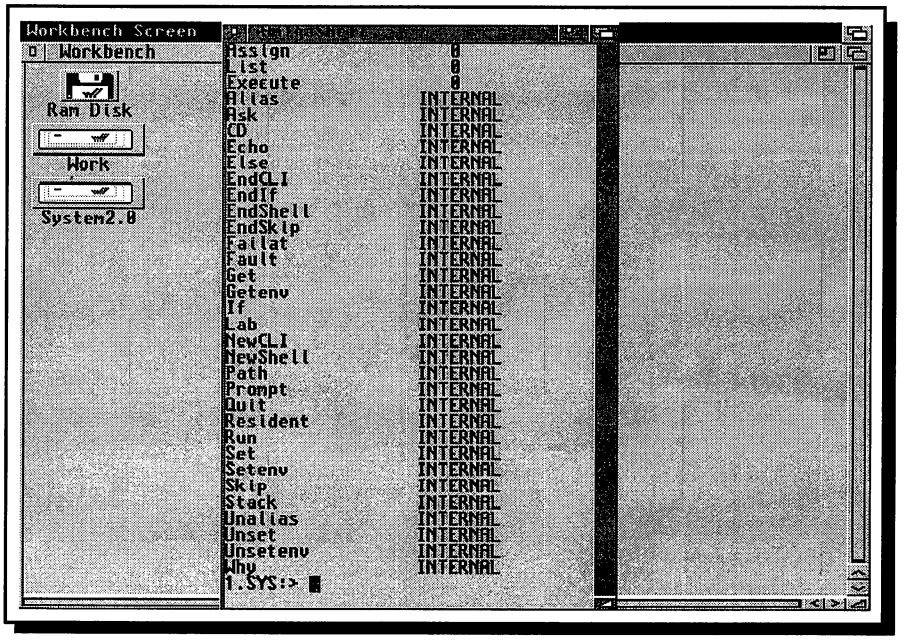


Figure 8-2 Resident Commands

The resident commands come in two flavors: commands you make resident with the Resident command and those built into Kickstart. The former have a Use Count associated with them; the latter are marked Internal in the resident listing.

Unless you change its operation using command arguments, the List command displays the name, size, protection bit settings, date and time stamp, and file note for each file and directory in the current directory. The name, of course, is the name of the file or directory, while size is the number of bytes of disk space required to store the file. Directories don't have a length; they are marked with a Dir in the size column. The protection bits are flags that you set with the Protect command. The date/time stamp lists the date and time you created or modified the file, and the file note is a comment that you append to a file using the Filenote command (see Chapter 9 for more on Protect and FileNote).

The List command's template is long, reflecting your many options:

```
DIR/M,P=PAT/K,KEYS/S,DATES/S,NODATES/S,
TO/K,SUB/K,SINCE/K,UPTO/K,QUICK/S,BLOCK/S,
NOHEAD/S,FILES/S,DIRS/S,LFORMAT/K,ALL/S
```

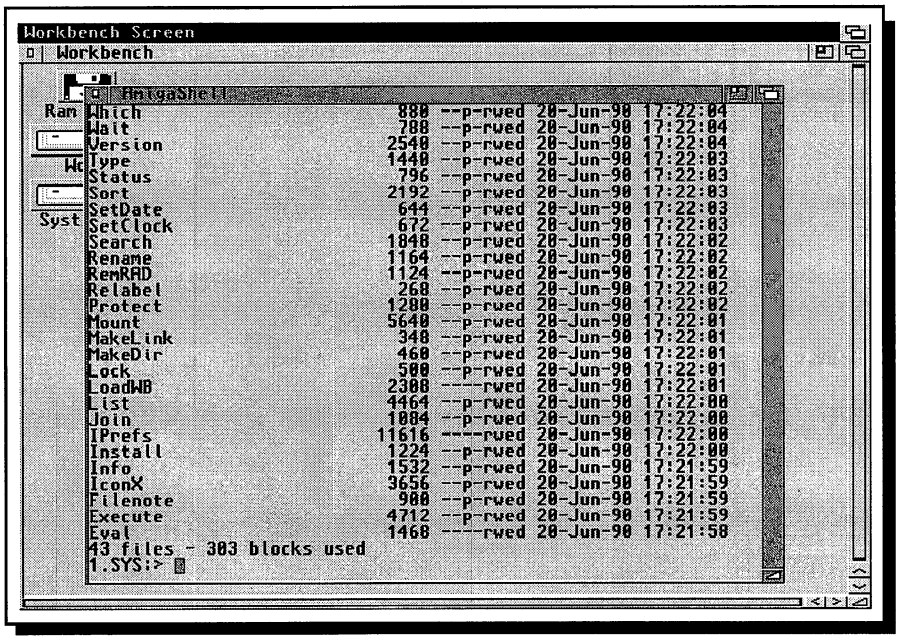


Figure 8-3 The List Command

Unlike Dir, the List command gives you the files' sizes, protection status, the time and date they were created or last modified, and any comments attached to them, as well as their names. At the bottom of the list, it displays the number of files and directories listed and the total disk space used, in blocks.

The DIR/M argument lets you list the contents of directories other than the current directory (which is the default) or, if you enter a filename, List will display information on that file only. You simply include the names of the files and directories you want on the command line.

List also supports pattern matching. The P=PAT/K argument indicates that you need to use the keyword P or PAT before a pattern for which you want to search, but this is not true. List supports patterns without the P and PAT keywords. List is restricted, however, in how it uses patterns. It cannot use them to match directories, but can be used to match the files within directories. For example, if you're in the root directory of your system disk, you can list the contents of the System and Utilities directory by entering:

```
LIST SYSTEM UTILITIES
```

Entering the following pattern:

```
LIST #?e#?
```

would produce only a list of files and directories within the current directory whose names contain the letter e, not a listing of all the directories whose names match the pattern. List uses pattern matching to decide which files — including directory files — it will display within a directory. It doesn't use pattern matching to determine which directories to list.

List also supports a substring search. Using the SUB/K argument, you can define a substring that a file or directory name must contain to be listed. For example:

```
LIST SUB yst
```

lists all the files with names that contain the character sequence yst. You can use pattern-matching wildcards in the substring, as well.

When you use the KEYS/S keyword, List displays the block number on the disk where storage of the file or directory begins. The BLOCK/S keyword displays the files' sizes in disk blocks used, as opposed to bytes.

The DATES/S keyword displays the date portion of the date/time stamp in DD-MMM-YY format, where DD is the day of of the month, MMM is a three letter abbreviation for the month, and YY is the last two digits of the year. In its default format, List uses the DD-MMM-YY format unless the file was created or modified within the last week, in which case it uses the day of the week (for example, Monday or Yesterday). DATES/S overrides this special treatment for recently created files. NODATES/S leaves off the date/time stamp entirely.

SINCE/K and UPTO/K let you list files based upon the date in their date-time stamps. SINCE/K limits the display to files created or modified since the date you enter, while UPTO/K specifies files created or modified up to the supplied date. For example, to see which files were created or modified since October 1, 1990, you enter:

```
LIST SINCE 01-oct-90
```

Note that in addition to entering dates in DD-MMM-YY format, you can also use the days of the week, Yesterday, Today, and Future with the SINCE and UPTO arguments.

TO/K lets you redirect List's output to a file. It functions identically to the output redirection operator. Thus, the following two commands perform the same function:

```
LIST >list_of_files
LIST TO list_of_files
```

List also lets you display files or directories only, by using the FILES/S and DIRS/S arguments, respectively. As a variation, QUICK/S displays the file and directory names only, without all the other information. Using the QUICK/S option is identical to using Dir, except that Dir alphabetizes its list, separates files from directories, and arranges filenames in two columns. The NOHEAD/S argument suppresses the heading that states the directory being shown and the date, plus suppresses the final message that details the number of files and directories listed and how many disk blocks they fill.

The ALL/S option works as it does with the Dir command, showing all files and directories in and below the current directory. It doesn't indent filenames to show the relative depth of files within the AmigaDOS file structure as it does with Dir, but it does provide the aggregate space used by all the files and directories it lists.

The last option recognized by List is LFORMAT/K, which lets you format the output of the List command. Specifically, it lets you embed the output in a string of characters. For example, if you enter:

```
LIST C: LFORMAT The AmigaDOS command %S.
```

your output will look similar to that in Figure 8-4. Here, LFORMAT/S caused List to embed its output in a string by substituting its output for the %S substitution operator. When using LFORMAT, the NOHEAD and QUICK options are automatically invoked.

Why would you use LFORMAT/S? Well, it is a great way to automatically generate script files. For example, if you wanted to add the same extension to all the files in a directory — such as appending a .pic extension to a directory full of IFF picture files — you couldn't use a single Rename command because it does not support pattern matching. Instead, you could create a script file with one Rename command for every file in the directory. If the picture directory were named Work:Pictures, you would create the script file by entering:

```
LIST >rename.script WORK:PICTURES LFORMAT RENAME %S%S
%S%S.pic
```

This command creates a file named rename.script. Each line in the file is a command that renames one of the files in the Work:Pictures directory by appending a .pic extension to the filename. To rename the files, you then enter:

```
EXECUTE rename.script
```

(For more about scripts and the Execute command, see Chapter 10.)

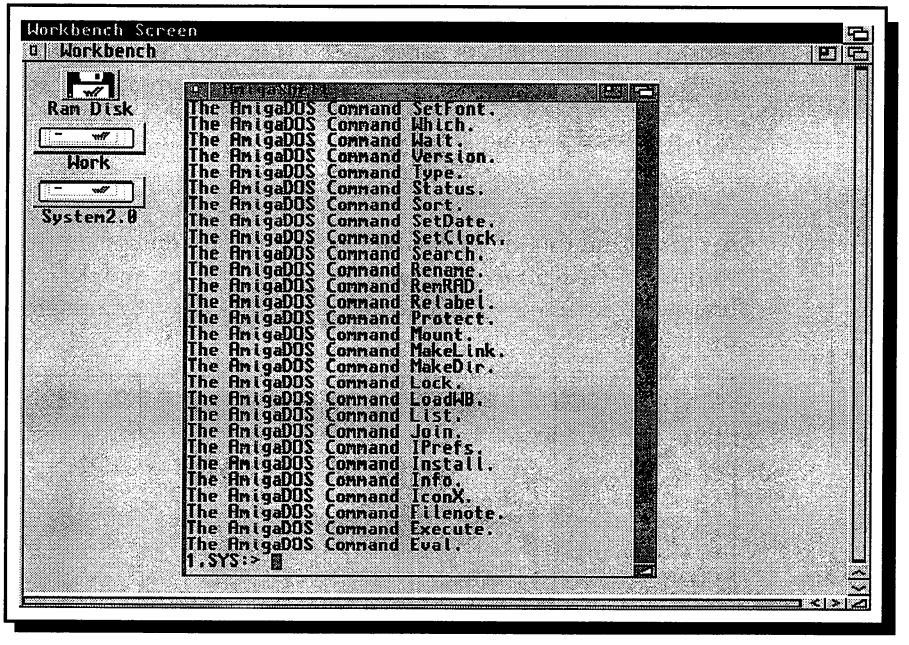


Figure 8-4 The LFORMAT Argument

LFORMAT lets you embed the output of the List command in a string of text. By redirecting the output of the command to a disk file, you can easily make command scripts that work on every file listed.

Note the multiple appearances of the %S substitution character in the example command. Exactly what is substituted for %S depends upon how many of these characters are in the LFORMAT string. List replaces one %S with the filenames; with two %S characters, List replaces the first one with the path and the second with the filename. With three %S characters, List substitutes the path for the first, and the filename for the second and third. With four characters, as in the example above, List substitutes the path for the first character, the filename for the second, the path for the third, and the filename for the fourth. Thus, each line in the rename.script file created above has the form:

RENAME *path/filename path/filename*.pic

Assign: Assigning Logical Directories

In the last chapter, you learned about logical directories that AmigaDOS uses to ensure programs always know where to find the AmigaDOS commands or

the disk-based libraries. You also learned how to use logical directories as shortcuts in place of long pathnames. Listing and modifying the current assignments active on your system is the Assign command's task.

When entered without arguments:

```
ASSIGN
```

the command lists all the logical names active on the system (see Figure 8-5), showing the assignments given volumes, directories, and devices. In effect, it lists all the items on your system that you can access by a single name followed by a colon.

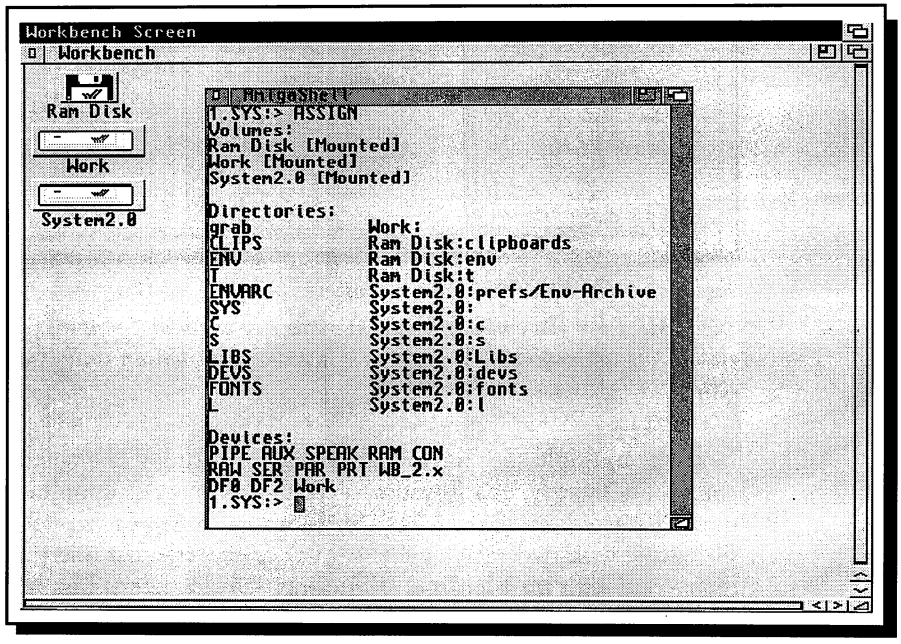


Figure 8-5 The ASSIGN Command

Assign lists all the disk volumes, logical directories, and devices active on your system. Here, it shows the basic assignments active after you boot an Amiga 3000. Assign also lets you make your own logical assignments to volumes and directories.

You don't change volume or device names using Assign. You change the volume names shown in the Assign listing via the Relabel or Rename commands (see Chapter 9) or the Icon menu's Rename item. You add volumes as you put disks in your drives, and subtract them as you remove disks. You add devices using the Mount command. Assign is primarily concerned with adding, subtracting, and changing logical directories.

The Assign command's template is:

```
NAME,TARGET/M,LIST/S,EXISTS/S,DISMOUNT/S,
DEFER/S,PATH/S,ADD/S,REMOVE/S,VOLS/S,
DIRS/S,DEVICES/S
```

To make a logical assignment, you first enter the name you want to use, followed by the name of a physical directory. For example, to assign the name Pics: to the Work:Pictures directory, enter:

```
ASSIGN PICS: Work:Pictures
```

The logical name must end with a colon and be separated from the physical pathname by a space. From the template, you'll note that the TARGET argument has a /M modifier. This means that you can assign the same logical name to different physical directories. For example, if you have another directory on Work named Graphics that also has a lot of picture files, you might want to use the following:

```
ASSIGN PICS: Work:Pictures Work:Graphics
```

Whenever you refer to Pics: from AmigaDOS or a program, AmigaDOS will check the first directory listed; If it can't find the required information, it checks the second directory. If you had already made one assignment for Pics:, you don't have to reenter that assignment to add another directory. With Pics: already assigned to Work:Pictures, you can add the Work:Graphics directory using the ADD/S argument:

```
ASSIGN PICS: Work:Graphics ADD
```

Contrary to Commodore's documentation, the Assign command does not take a SUB argument that lets you subtract one of the physical directories assigned to a logical directory. You can, however, erase an assignment by making a new one or by using the REMOVE/S keyword. For example, the following erases the assignment for Pics:.

```
ASSIGN PICS: REMOVE
```

You can also remove volumes and devices from the assignment list using the DISMOUNT/S argument. Removing a disk drive from the list of devices means that you'll be unable to access the disk from the Shell, although it will still be available through Workbench. Using DISMOUNT/S doesn't save any system resources. Refrain from using it.

When you assign a name to a physical directory, Assign tries to make sure that the directory exists. If the directory is on a disk that is not mounted, Assign asks you to insert the disk so that it can confirm the directory's presence. To

suppress this check, use the DEFER/S option to keep AmigaDOS from looking for the physical directory until you actually use the logical name.

By default, the Assign command also expects you to enter the pathname of a directory when you assign a logical name to it. You can enter either a relative or absolute pathname, but it expects a unique pathname for every assign. You can change this default using the PATH/S option. In this case, you enter the path for a directory without specifying its volume. For example, if you enter the following:

```
ASSIGN PATH C: :C
```

AmigaDOS will look for its commands on any disk that has a C: directory — rather only on the disk that you used to boot your system. This is especially helpful to people with one disk drive who have to swap their Workbench disk with another disk in order to run a program.

The EXISTS/S argument is useful for in script files. It determines whether a logical assignment already exists. If the assignment doesn't, the command returns with a warning. (For more on return codes and how to test them, see Chapter 10.) When you use the EXISTS/S argument interactively from the Shell, it either tells you the directory a logical name is assigned to or it indicates that the logical directory doesn't exist. Note that this argument checks for the existence of a logical name, not for the use of a physical directory.

The final four arguments deal with the information the Assign command returns. The LIST/S keyword outputs the same list as using Assign without an argument; you therefore never need use it. The VOLS/S argument causes Assign to list the volumes, while DIRS/S and DEVICES/S output the assigned directories and mounted devices, respectively. Thus, the following three commands are equivalent:

```
ASSIGN
ASSIGN LIST
ASSIGN VOLS DIRS DEVICES
```

Remember that none of the logical directory assignments are carved in stone. You can change the assignments of the C: and L: directories as easily as you can for logical directories you create yourself. You must be sure that any directory you assign to one of the system directories can handle that directory's functions. For example, if you assign C: to My_Disk:AmigaDOS/Commands then you'd better have some commands in the latter directory.

One further note about the Assign command: It places restrictions on the names you can give disk volumes. If you name a disk C:, for example,

AmigaDOS may confuse it with the logical directory C:. The same goes for the other logical names active on the system. You should avoid using any names for disks that conflict with the logical names active on your system.

Resident

As I noted above, AmigaDOS commands are stored in two places; the C: directory of your system disk and in Kickstart 2.0. You get a listing of the former by entering:

```
DIR C:
```

For a listing of the latter, type:

```
RESIDENT
```

Commands in the list followed by a Use Count are those that you (or your Startup-sequence) made resident. Commands marked Internal are always present in Kickstart.

Commands listed by Resident are always available to you. Unlike the C: commands, which you may have to swap disks to use if the disk containing the C: directory isn't in a drive, resident commands are always on hand, because they have, in effect, been incorporated into AmigaDOS. Resident not only lists such commands, but also lets you make other C: commands resident.

To be resident, a command should be pure; that is, it should be re-entrant and reexecutable. A re-entrant command is one that can be interrupted at any point and can later continue execution as if nothing had happened. A reexecutable command isn't changed when it is executed; no matter how often you run it, it performs identically as it did the first time you ran it. Both these qualities are vital for commands running in a multitasking system. Any resident command must be ready to be executed from more than one Shell at the same time. If you run the same command from the C: directory in two different Shells, each Shell loads its own copy of the command. If the command is resident, each Shell is executing the same code at the same time. This requires pure code.

The template of the Resident command is:

```
NAME,FILE,REMOVE/S,ADD/S,REPLACE/S,PURE=FORCE/S,SYSTEM/S
```

To make a command resident, you have to specify its complete pathname. For example, to make the Copy command resident, you enter:

```
RESIDENT C:COPY
```

If Copy is pure, the system adds it to the resident commands and subtracts the amount of memory that Copy occupies from the available memory. The filename Copy will appear in the resident list.

An interesting aspect of the Resident command, the NAME argument lets you give a command a different name from the one it has on disk. For example, if you enter:

```
RESIDENT George C:DIR
```

you will make the Dir command resident under the name George. Now, when you enter:

```
George
```

you get a listing of the current directory. This isn't a terribly useful feature, but it can be helpful if you're used to other command names on other operating systems.

To remove a command from the resident list and free up the memory it uses, you use the REMOVE/S option:

```
RESIDENT George REMOVE
```

Note that you can't remove a command if it is being used by another Shell. Check the Use Counts in the resident list for a command's status. A command's Use Count must be zero before it can be removed. If you remove an internal command, the system will treat it as if it doesn't exist, although, of course, it is still present in Kickstart. It will show up again only when you reboot your system. Removing internal commands is useful if you want to use a disk-based command that has the same name.

The ADD/S and REPLACE/S options are redundant. ADD/S says to add a name to the resident list and is the default setting of the Resident command. Likewise, if you specify a name that is already in the resident list, Resident automatically replaces the old name with the new, without the REPLACE keyword.

If you try to make a command resident that isn't pure, you get a message saying that the object is not of required type. To override this objection, use the PURE=FORCE/S option. For example, the Eval command in C: is not pure, but you can make it resident by entering:

```
RESIDENT C:EVAL PURE
```

or, because they are equivalent:

```
RESIDENT C:EVAL FORCE
```

The question is, do you want to make impure commands resident? If you make commands that are not re-entrant and reexecutable resident, you're asking for software failure. Avoid forcing any impure command to be resident.

How do you tell if a command is pure? Just look at the output of the List command. A pure command has its p (for Pure) protection bit set. If you enter:

```
LIST C:
```

all the commands with p as one of their protection bits are pure. Avoid the temptation to use the Protect command to set the pure bit on a command that isn't. Setting the bit doesn't make a command pure — only proper development and extensive testing can ensure that. Unless you are an accomplished programmer and software tester, don't mess with the p protection bit or force nonpure commands to be resident.

The final option for the Resident command is SYSTEM/S. When used alone, this argument lists those pieces of code (they aren't always commands) that have been added to the system software. For example, enter:

```
RESIDENT SYSTEM
```

Figure 8-6 shows the results of this command. Unlike normal resident and internal commands, files labeled System can't be removed from the resident list.

When used with a command pathname, the SYSTEM/S option adds the file to the system list. Commands you add to the system list stay resident until you reboot. You should avoid using this option, which is intended primarily for important system software.

If you have a lot of memory, you should make any commands you use often resident. Because resident commands don't have to be loaded from disk, they execute much faster. If you have a single-drive system, you should also consider making the more popular commands like Dir and Copy resident to free you from disk swapping. The default Startup-sequence for AmigaDOS 2.0 already makes Assign, List, and Execute resident. Resident lets you add to these three.

The CD Command

In Chapter 7, you saw how you could move about the AmigaDOS file structure simply by entering the name of a directory. For example, if you are in the root directory of your system disk, you can move to the Utilities directory by entering:

```
UTILITIES
```

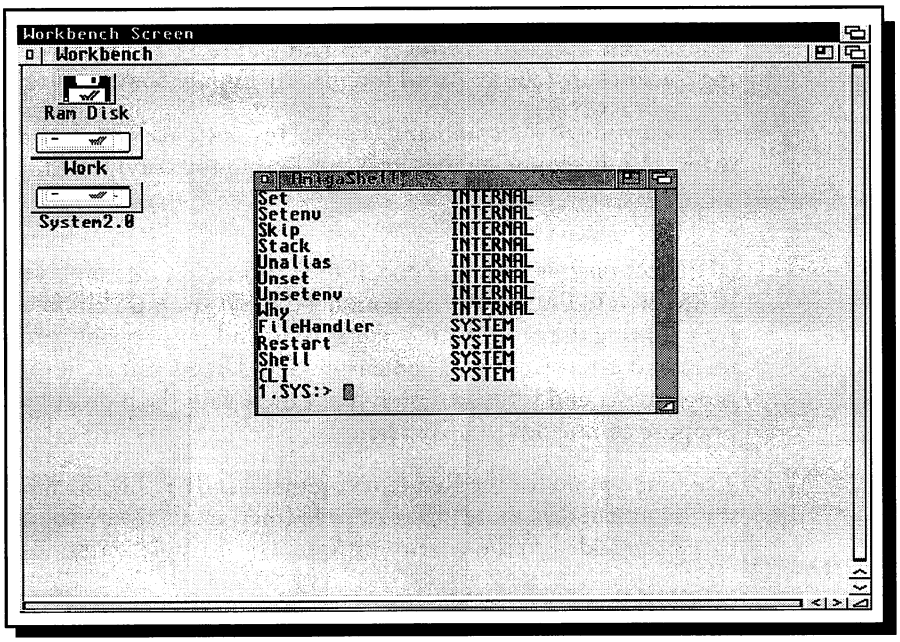


Figure 8-6 The SYSTEM Option

Entering the SYSTEM option with the Resident command lets you see those resident files that you cannot remove. They appear after the listing of the Internal commands.

This capability is new to AmigaDOS 2.0. In earlier versions, you had to use the CD (Current Directory) command to change directories, as in:

```
CD UTILITIES
```

CD has one other function, to list the current directory. Thus, when you enter:

```
CD
```

after moving into the Utilities directory, AmigaDOS will respond:

```
Workbench2.0/Utilities
```

Because this information is available in the prompt string as well, CD is pretty much obsolete.

Type

Unlike commands that display lists of files, the Type command is useful for examining the contents of both programs and data files.

The template of the Type command is:

FROM/A/M,TO/K,OPT/K,HEX/S,NUMBER/S

By default, Type displays the contents of a file you indicate to the Shell window. Thus, by entering:

TYPE S:Startup-sequence

you see the contents of your Startup-sequence file. Because the FROM argument is required — it has a /A modifier — you'll get an error message if you enter Type without specifying a filename. Of course, the /M modifier means that you can type multiple files by entering their names on the command line.

The TO/K argument lets you redirect Type's output to a file or AmigaDOS device. For example, entering:

TYPE S:Startup-sequence TO PRT:

outputs the file to your printer. Redirecting the output of Type to another file in effect copies the original file. Note that unlike earlier versions of AmigaDOS, the TO keyword is now required in order to redirect the output of Type to another file.

The OPT/K argument provides two options, H and N, which are equivalent to the HEX/S and NUMBER/S arguments, respectively. If you try to type a file that contains binary data — a program file is a good example — you wind up with a lot of strange characters on your screen. To properly view a binary file, such as the Preferences Palette editor, you should enter either:

TYPE Sys:Prefs/Palette OPT H

or

TYPE SYS:PREFS/Palette HEX

Both of the above commands produce the output that is partially shown in Figure 8-7. Each line of the output consists of a hexadecimal number that shows the offset of the line from the beginning of the file, the contents of 16 bytes of the file, and the ASCII interpretation of those bytes. ASCII is a standard way to represent alphanumeric characters with numbers. Because most of the contents of a binary file usually doesn't translate into ASCII, this part of the output may be gibberish.

The OPT N and NUMBER keywords output files with line numbers, but are not available with the hexadecimal output.

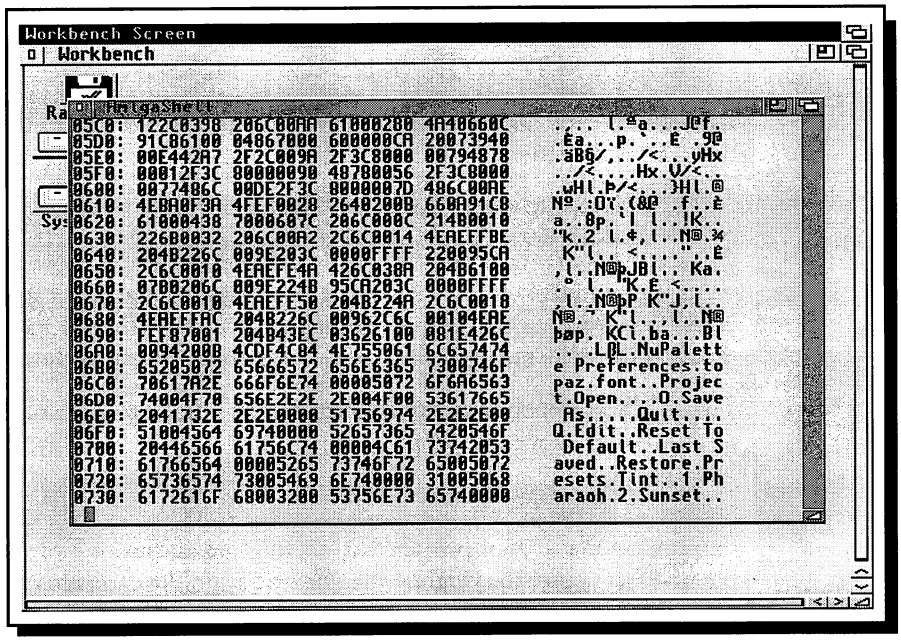


Figure 8-7 Typing a Binary File

This is what a binary file — in this case, the Preferences Palette editor — looks like when output with the HEX option. Note that the hexadecimal digits are 0-9 and A-F. The ASCII interpretation to the right of the hexadecimal digits often won't make sense when you're dealing with a program file.

Remember that, as with any command, you can halt output by pressing any key and resume it by pressing the Backspace key or CTRL-Q. You can abort the output by pressing CTRL-C.

Search: Looking Inside Files

Type simply outputs the contents of files; it is useful for examining a short file, but it can be tedious if you have to look through a large number of files for a particular piece of data. That is the province of the Search command.

Search looks through the files you specify for a particular string of characters. If successful, the command reports the location of the string in the file. For example, if you have a number of word-processing files in a directory and you can't remember which one has the information on your department's 1991 budget, you can automatically search all the files for the characters 1991 Budget.

The template for Search differs in important ways from the one given in Commodore's documentation. It is:

```
FROM/A/M,SEARCH/A,ALL/S,NONUM/S,QUIET/S,
QUICK/S,FILE/S,PATTERN/S
```

FROM/A/M lets you indicate the files you want to search. You can search a single file, multiple files, or multiple directories, or use pattern matching to determine which files to search. The SEARCH/A argument is the string you want to find. It can contain pattern-matching wildcards, but you need not worry about capitalization because Search ignores case. If the search string includes a space, you must enclose the entire string in double quotes. Thus, to search all the files in your Work:WordProcessing drawer for the sequence Budget 1991, you enter:

```
SEARCH WORK:WORDPROCESSING Budget 1991
```

The Search command responds by listing all the files it has searched, and all the occurrences of the string in those files. It prints both the line number and the contents of all lines in the files containing the string.

Search gives you three output options and two search options. The NONUM/S argument lists the lines that match the search string without using line numbers. QUICK/S outputs the files that contain matches for the search string, without displaying the matching lines. QUIET/S lists only the files that produce a match.

If you specify a directory for your search, the ALL/S option not only lets you search that directory, but also look through all the files and directories below the specified one in the AmigaDOS file structure. The FILE/S options let you indicate that the string you're searching for is a filename and not a text string within a file. Thus, to search your entire Workbench disk for a file named Clock, you enter:

```
SEARCH Workbench2.0: Clock FILE ALL
```

Search then does a quiet search of your entire disk, listing the complete pathnames of all files named Clock. Note that you can't search for directories using FILE/S.

For my money, the ability to find a filename anywhere on a disk is the most useful function provided by Search. I never used this command before researching this book; now I use it all the time to find files that I've lost on my hard drive.

Dealing with Processes

Up to this point, the commands in this chapter have dealt primarily with files and directories. The commands below deal more with process than with files. In many cases, the scope of these commands is limited to the Shell in which you are currently working.

The commands in this section are Alias, Break, ChangeTaskPri, EndCLI, EndShell, Fault, LoadWB, NewCLI, NewShell, Path, Prompt, Run, SetFont, Stack, Status, UnAlias, Which, and Why.

In Amiga parlance, each Shell is a process. The commands in this section deal with individual processes.

Alias and UnAlias

Alias is a simple yet powerful command that lets you tell the Shell to treat one string of characters as another. As an example, examine the alias names set up by the standard Shell-startup file (see NewShell below). Enter:

```
ALIAS
```

The Shell responds:

```
emacs memacs
xcopy copy clone
```

The left column contains the names the Shell looks for in the command position on the input line. Alias only substitutes for the first *word* you enter on the command line (see Chapter 7). The right column shows what the system sends to AmigaDOS when the Shell parser encounters the sequence in the left column. Thus, when you enter emacs on the command line, the Shell actually sends Memecs to AmigaDOS. Thus, you can start up the MicroEMACS editor by entering emacs instead of its actual filename, which is Memacs.

Who cares? Well, if you use EMACS on another system, you might want the convenience of accessing it with same name on your Amiga. True, you could simply rename the Memacs file, but then programs that call Memacs wouldn't be able to find it. Using an alias, you can call the file anything you want without interfering with the ability of programs or other users to find it.

Setting up an alias is easy. The template for the command is:

```
NAME,STRING/F
```

NAME is the new name the Shell will accept in place of the STRING. Remember, the /F modifier means that you can include spaces in the string without using quotes as long as the string is the last item on the command line. Because you can not be sure that what you enter here will always be the last thing on every command line that uses the alias, you should enclose the STRING argument with quotations marks if it contains spaces.

For example, let's assume that you don't like the default output of the List command, yet you're tired of entering all the keywords it takes to get the output you want. You can create an alias for List that includes all the parameters you want. To see what I mean, enter:

```
ALIAS li LIST NODATES BLOCK KEYS
```

Now, whenever you enter li, you get the same results as entering List with its three arguments. Alias is a great way to change defaults you don't like without having to reprogram a command. It's also a great way to cut down on your typing. For example, if you use the alias CPI for the command ChangeTaskPri, you'll save yourself a lot of typing. You'll also ensure that any program that calls ChangeTaskPri will be able to find it — something it couldn't do if you simply renamed the command.

UnAlias is the opposite of Alias; it removes a name from the alias list. Its template is:

```
NAME
```

If you enter UnAlias without entering a name, it simply lists the aliases active in the current Shell.

ChangeTaskPri: Changing Priorities

As a multitasking operating system, Amiga OS is performing a constant juggling act. To create the illusion that all active processes are executing simultaneously, the operating system gives each process a set amount of time on the CPU. When one's time is up, the operating system swaps out that process and swaps in another. This can happen hundreds of times per second.

The **ChangeTaskPri** command lets you have some control over the relative amounts of time different processes have access to the processor. A process with a high priority will have more time on the processor than one with a low priority, and will thus execute faster. The template for ChangeTaskPri is:

```
PRI=PRIORITY/A/N,PROCESS/K/N
```

PRI (or PRIORITY) is a number from -128 to 127; the higher the number, the higher the priority. The default setting for all processes is 0. To give a process greater than average access to the CPU, you enter a positive number. Entering a negative number assigns a relatively low priority. Commodore recommends setting priorities between -4 and 4. Below -4, the process may take too long to execute. Above 4, it may interfere with vital operating system processes.

PROCESS/K/N indicates the number of the process whose priority you wish to change. The process number is in front of the name of the current directory in the prompt string. If you leave off the PROCESS argument, you are by default setting the priority of the current Shell.

For example, to change the priority of process 4 to 2, you enter:

```
CHANGETASKPRI 2 PROCESS 4
```

Process numbers are also available using the Status command.

NewCLI and NewShell: Starting and Ending Shells

From Workbench, you open a Shell by opening the Shell project in the root directory of your system disk. You can also open a Shell from a Shell, using the NewCLI and NewShell commands. These two commands are equivalent; the two names are a carryover from earlier versions of the operating system when a Shell and a CLI were two different things. To keep the discussion simple, I'll describe NewShell only.

When you enter NewShell without any arguments, it opens a Shell window at the top of your screen that inherits the stack and path settings of the Shell from which it was spawned. NewShell's template is:

```
WINDOW,FROM
```

As you can see, neither the WINDOW or the FROM arguments are required (no /A modifiers); in fact, they are hardly ever used. The WINDOW argument lets you use an alternate console device besides the standard CON: and define the dimensions and location of the Shell window. The WINDOW argument takes the following form:

console device:x/y/w/h/title

console device: A console device lets an Amiga window emulate the functions of a video display terminal. The standard Amiga console device, CON:, is built into Kickstart and automatically mounted when you boot your system. If you wish to use an alternate console device, you must move its handler to

the L: directory, provide a mountlist entry, and mount the device (see the Mount command in Chapter 9). With the improvements made to CON:, the reasons for seeking out an alternate console handler from the public domain aren't as compelling as they were for users of Amiga OS 1.2.

x: Indicates the number of pixels from the left edge of the screen where the top-left corner of the Shell window will appear.

y: Sets the number of pixels from the top of the screen where the top-left corner of the window will appear. In conjunction with *x*, it defines the location of the top-left corner of the window.

w: Defines the width of the Shell window in pixels.

h: Defines the height of the window in pixels.

title: A string that indicates the name that will appear in the Shell title bar. If the title contains a space, the entire WINDOW argument must be enclosed in quotation marks.

Curiously, any Shell you open using a WINDOW argument to override the defaults comes up without a close gadget.

The FROM argument lets you specify a startup file for the Shell. A startup file is an AmigaDOS script file. Whenever you open a Shell, you can have it execute an AmigaDOS script. Normally, the script you execute will configure the Shell as desired. Without a FROM argument, NewShell executes the script Shell-startup, located in the S: directory. When you Type this file, you see:

```
alias xcopy copy clone
alias emacs memacs
;alias clear echo * *E[0;0H*E[J*
;alias reverse echo * *E[0;0H*E[41;30m*E[J*
;alias normal echo **E[0;0H*E[40;31m*E[J*
Prompt %N.%S>
```

These are the commands that NewShell automatically executes when it starts up a new Shell. You're already familiar with Alias, so you know what the first two commands do. (They're the reason that every Shell you start has the same two default aliases.) The next three lines, with Alias commands preceded by semicolons, are comment lines. Everything on a command line after a semicolon is ignored by AmigaDOS. Thus, these strange Alias commands don't take effect unless you use a text editor to delete the semicolons. Echo is a command you'll encounter in Chapter 10, where you'll also find out the meaning of the strange console-control characters. You'll learn about the Prompt command later in this chapter. When you learn how to create command scripts in Chapter 10, you'll be able to customize your own Shell startup files.

When you're through using a Shell created with NewShell or NewCLI, you can close it by simply entering EndShell or EndCLI. Neither of these commands takes any arguments. If you've launched a process using the Run command from a Shell, you can't close it until the process terminates. (See "Remote Control," later in this chapter, to break out of a process.)

LoadWB: Starting Workbench

In addition to starting other Shell processes, you can also start up the Workbench interface from the Shell. LoadWB, the program that creates and operates the Workbench interface, is an AmigaDOS command that resides in the C: directory of your system disk. LoadWB doesn't take any arguments. If you execute it while the Workbench is active, it does nothing — you can't have two Workbench screens active at once on your machine. If you ever quit Workbench, or due to some error, drop out of the Startup-sequence script before you get to the LoadWB command, you can enter the command manually to start up the Workbench interface.

Path: AmigaDOS Pathways

When you press Return after entering a command into the Shell, the Shell sends the first word it parses on the line alone to AmigaDOS. The first word, of course, is the command name. To execute the command, AmigaDOS must be able to find and load the command. Using the Path command, you can tell AmigaDOS where to search for its commands.

If you enter a command you've made resident or that is internal, AmigaDOS doesn't have to search; these commands are immediately available. With all other commands, however, AmigaDOS must find and load the commands before executing them. AmigaDOS first looks for commands in the current directory. If unsuccessful, it next searches all the directories in the specified search path. After checking the path, it looks in the C: directory. If, after all this, AmigaDOS hasn't found the command you entered, it will report object not found. (You'll also get this message if you spelled the name of the command wrong.) AmigaDOS uses the same path to search for commands and command scripts.

To show or alter the command search path, you use the Path command. It has the following template:

```
PATH/M,ADD/S,SHOW/S,RESET/S,QUIET/S,REMOVE/S
```

To see the current search path, enter:

```
PATH
```

The system returns a list of all the directories that AmigaDOS searches when you enter a command or command script. If the path includes a directory that is on an unmounted disk, you'll be prompted to insert the disk into a drive. Entering the Path command without arguments is equivalent to using the SHOW/S argument.

To add directories, you enter their pathnames on the command line, separated by spaces. (You can also include the ADD/S argument but, like SHOW/S, this is the default, rendering the keyword unnecessary.) Thus, to add the Prefs and Prefs/Presets directories to your search path, you enter:

```
PATH Sys:Prefs Sys:Prefs/presets
```

When you next enter Path, these will show up in the listing. If you enter a directory on a disk that is not mounted, you'll be prompted to insert the disk.

PATH settings are inherited from one Shell to another. When you first open a Shell after booting it inherits the path established by the Startup-sequence. If you change this path, any Shells you spawn will inherit the new path.

You can remove a directory from the search path by using the REMOVE/S argument. For example, to remove the two directories we just added, you enter:

```
PATH Sys:Prefs Sys:Prefs/presets REMOVE
```

To remove all directories from the search path except for the current directory and C: (which can never be removed), you enter:

```
PATH RESET
```

The final Path argument is QUIET/S, which is supposed to keep AmigaDOS from prompting you to enter a disk when your path contains directories from unmounted disks. In the version of Amiga OS I worked with, this option simply suppressed all output from the Path command. Obviously, it needs a little work.

Which

The Which command is a cousin to Path. When you enter a command name with Which, it returns the complete pathname to the command. If the system cannot find the command in the current search path, Which returns a warning

(return code equals 5) that you can test for in a script file. The template for Which is:

FILE/A,NORES/S,RES/S,ALL/S

The FILE/A argument is the name of the command you wish to find. If you include the NORES switch, Which will not search the resident list for the command. (Internal commands are considered resident by Which.) The RES switch searches the resident command list only.

Normally, Which quits after it finds a file that satisfies your search. If you use the ALL/S argument, the command searches every directory in the path, regardless of how many hits it gets in its search. For example, to find every place the command Copy exists in the search path, you enter:

WHICH COPY ALL

The Prompt String

Every Shell window has a prompt string; those characters to the left of the cursor that normally indicate the process number of the Shell and the current directory. This standard prompt comes from the **Prompt** command in the Shell-startup command script.

The Prompt command template is, simply:

PROMPT

which can be any string of characters you wish. For example, if you enter:

PROMPT Go Red Sox! >

that will be your new command-line prompt.

The Prompt string supports two substitution characters. If you put a %N in the Prompt string, the Shell outputs its process number instead. If you use %S, the Shell puts the path of the current directory in the prompt. Thus, the default prompt is actually defined by the simple command:

PROMPT %N.%S>.

Prompts are inherited when you create a NewShell. If you enter Prompt without any argument, it resets the prompt to the default.

Run: Background Processes

Normally, when you execute a program — such as an AmigaDOS command — from the Shell, you can't use the Shell until the program finishes execution. Most commands normally execute quickly, but sometimes you'll start a lengthy program that will lock you out of the Shell until it finishes. To avoid this conflict, you can run such programs in a background process using the Run command.

Run has no template; when used with another command, it runs that command in the background, freeing the Shell for your use. A background process doesn't open its own input and output window, although the program running in the process can.

For example, if you want to have your Shell free while you open the Ed editor, you enter:

```
RUN ED myfile
```

Ed will open its window, and you'll be free to continue working in the Shell. The only restriction is that you can't close the Shell until Ed finishes running. To break the connection between the Shell and the background process completely, you redirect the output of Run to the NIL: device:

```
RUN >NIL: ED myfile
```

Using the Run command, you can string many AmigaDOS command lines together using the addition operator (+). For example:

```
RUN DIR >SYS:dir.list DF0:+
TYPE TO SYS:start_file S:Startup-sequence+
RESIDENT >PRT:
```

will execute all three commands automatically. Note that because none of the commands open their own window, I had them redirect their output from the Shell window so that they wouldn't interfere with my continued use of the Shell.

The Stack Command

Every Shell has a stack, an area of temporary storage that commands use during execution. For example, when a command is interrupted to give another process time on the processor, the current state of the command is saved to the

stack and used to restart the command when the processor is again free. The **Stack** command lets you see and set the size of the stack for a Shell process.

When you enter Stack without an argument, it returns the size of the stack allocated to the current process. The standard Shell stack is 4,096 bytes long. To change the value, simply enter Stack, followed by the new size in bytes. Thus, to up the stack to 50,000 bytes, enter:

```
STACK 50000
```

Most AmigaDOS commands work fine with 4,096 bytes, but Sort may need more. Some commercial programs require a larger stack; you should check their documentation. Remember that unused memory in the stack is wasted and not available. Unless you run into recurring problems with a program, keep the stack at the default size.

SetFont: Changing The Shell Font

Another way you can customize the appearance of the Shell window is to change the font used to render text. Normally, the Shell uses the system default font set with the Preferences Font editor. You can override this default using the SetFont command, whose template is:

```
NAME/A,SIZE/A,SCALE/S,PROP/S,ITALIC/S,BOLD/S,UNDERLINE/S
```

The NAME/A and SIZE/A arguments are the font you wish to use in the Shell window. This font must exist in your Fonts directory. For example, to change the font used to Courier 24, you enter:

```
SETFONT Courier 24
```

The three style arguments, ITALIC/S, BOLD/S, and UNDERLINE/S, let you set the type style for the font. You can use all three styles in the same window if you want.

The CON: device doesn't handle proportional fonts very well on the command line, so you may want to stick with fixed width-fonts for the Shell. Specifically, when you use a proportional font, the cursor is often narrower than many of the characters you enter on the command line. Thus, it often chops off about half of each character, rendering your input unreadable. Commodore included the PROP/S argument apparently to let you indicate that you were specifying a proportional font so the Shell could adjust. From what I've seen, the PROP/S argument has no affect on how the Shell handles proportional fonts. If you want to use a typeface other than Topaz or Courier in your Shell

window, be prepared for unreadable input. Proportional output doesn't seem to be a problem.

The most interesting argument for SetFont is SCALE. It lets you scale an existing typeface up to 100 pixels high. For example, to create a Topaz 72 font, enter:

```
SETFONT Topaz 72 SCALE
```

The results are shown in Figure 8-8. The simple scaling method does not create sophisticated outline fonts, but it's better than nothing.

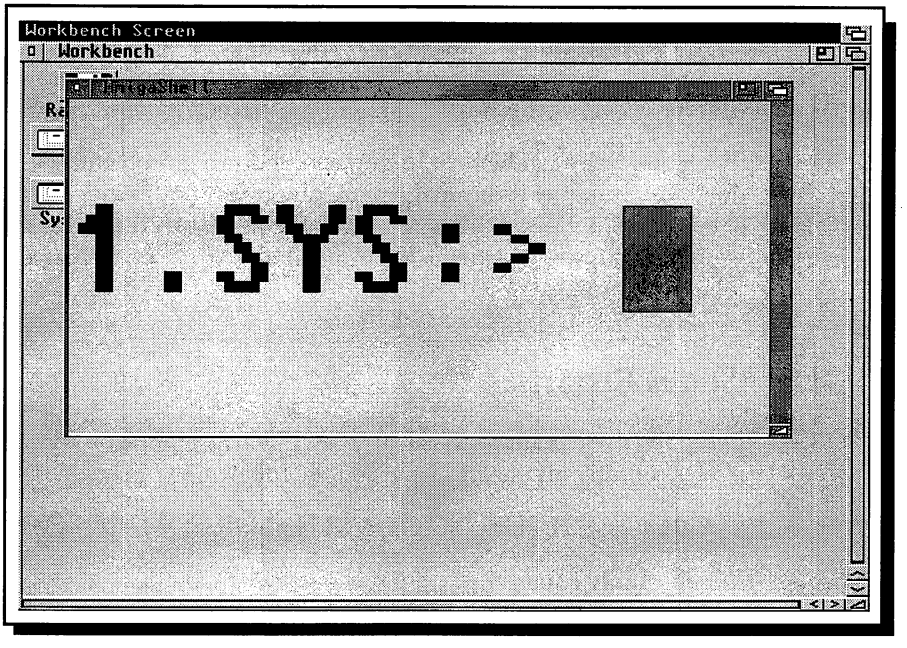


Figure 8-8 Scalable Fonts

Amiga OS 2.0 provides support for scalable fonts. This lets you define a pixel size for a typeface other than the ones contained in the Fonts: directory. Because Amiga fonts are bitmapped and not outline fonts, they can look pretty ragged when scaled to large sizes. This shows a 72-pixel Topaz font.

Fault and Why: Explaining Errors

Occasionally, when a command aborts execution because of an error, it prints an error number in the Shell window. For a slightly (and I do mean slightly) more enlightening explanation of the error, you can enter **Fault**, followed by

the error number. For example, if a command reports error code 116, you can enter:

```
FAULT 116
```

which brings the response:

```
Fault 116: required argument missing
```

You can enter up to ten error code numbers on a single command line.

Fault is related to the **Why** command. When a command fails, you can enter Why on the command line for an explanation of the failure. Most AmigaDOS commands now provide fairly verbose reasons for their failure, so the information provided by the Why command is often redundant.

Fault and Why can be helpful in tracking down a problem, but they are limited. For example, if they tell you an object was not found, you're still left with a lot of possibilities. Did you misspell the name of a file or use the wrong path? Appendix D, which explains the AmigaDOS error codes, may help you decipher what went wrong.

Status Report

Not terribly relevant to the average user, **Status** reports on the status of processes running on the CPU and has the following template:

```
PROCESS/N,FULL/S,TCB/S,CLI=ALL/S,COM=COMMAND/K
```

If you simply enter:

```
STATUS
```

the command produces a report like that in Figure 8-9, which lists the different processes running on your system and the commands currently loaded in them. This default report is also available using the CLI/S argument or its equivalent, ALL/S. By indicating a process number with PROCESS/N, you get a report of that process only. By entering a command name after the COM or COMMAND keywords, the system returns the number of processes running the command you indicate. Thus, to find the number of the process that is running the Sort command, you enter:

```
STATUS COM SORT
```

The command responds with the process number.

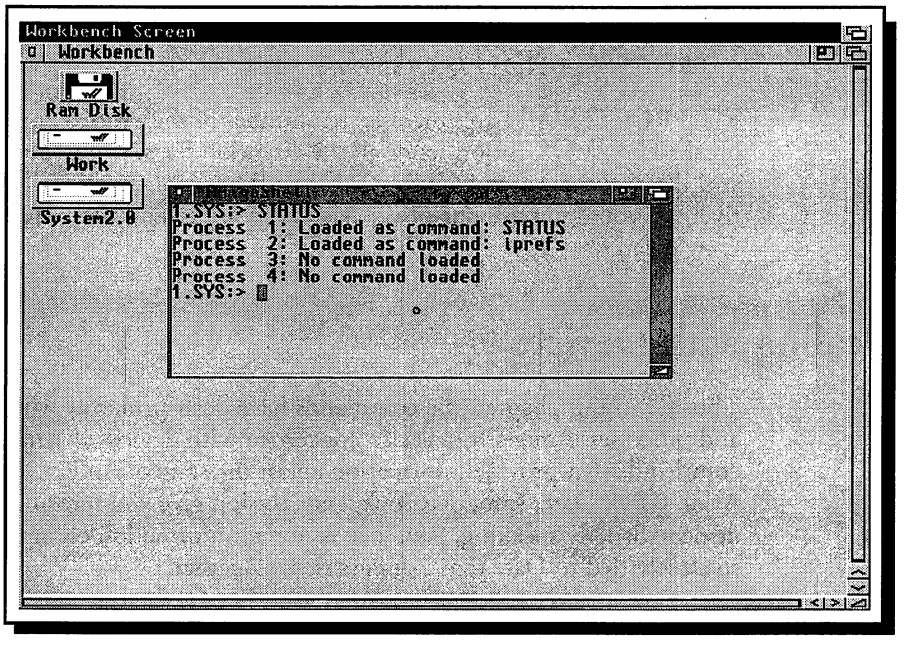


Figure 8-9 The Status Command
Status tells you which processes are active and which commands are running in them.

The TCB/S switch reports the task-control block information about all active processes, instead of the command information. The task control block gives the size of the stack (stk), information on the global vectors (gv), and the priority of the process (pri). The FULL/S option lists both task-control block and command information about the processes.

Unless you're a programmer, the information provided by the Status command is not very useful.

Break: Remote Control

Once you've used the Status command to learn the numbers of the different processes running on your system, you can use the Break command to abort any of them. Break lets you set the attention flags (CTRL-C, CTRL-D, CTRL-E, and CTRL-F) of any process on the system. CTRL-C aborts a command, while CTRL-D aborts a command script. The other two are currently undefined.

The template for the Break command is:

PROCESS/A/N,ALL/S,C/S,D/S,E/S,F/S

The PROCESS/A/N argument is the number of the process you want to break. The ALL/S option sends all the attention flags, while the C/S, D/S, E/S, and F/S arguments send CTRL-C, CTRL-D, CTRL-E, and CTRL-F, respectively. If you don't use one of the flags as an argument, the system sets CTRL-C by default.

Dealing with Devices

Thus far in this chapter, the commands have dealt primarily with directories and processes. AmigaDOS is also concerned with devices and has a number of commands that provide information about the state of AmigaDOS devices. Avail, CPU, Date, Info, SetClock, and Version give you information about devices. Strictly speaking, SetClock is not an information command, but it is so closely tied to Date that I discuss them together.

Info: Information Please

The Info command gives you information about the disk devices attached to your system. For a report, you simply enter:

INFO

The top half of the output (see Figure 8-10) consists of eight columns of information about the disk devices — hard disks, hard-disk partitions, RAM disks, and floppy disks — mounted on your system. The columns are Unit, Size, Used, Free, Full, Errs, Status, and Name.

Unit: The unit name of the physical name of a particular disk or partition. If a drive is mounted but has no disk in it, you'll see a message saying no disk present.

Size: Size lists the storage capacity of all mounted disks, in kilobytes (K) or megabytes (M).

Used: The Used column tells you how much storage space is already allocated to storing files. As you create more files on a disk, its Used figure increases.

Free: Lists the amount of unallocated storage space on a disk.

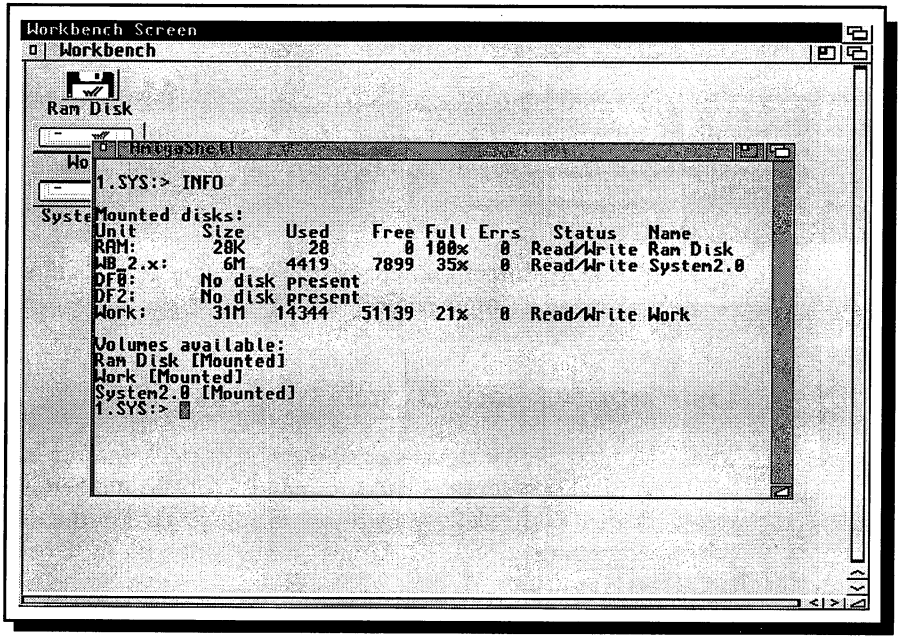


Figure 8-10 Disk Information

The Info command lets you know what disks and volumes are available to you and how much more data they can store.

Full: This column lists the amount of used space on the disk as a percentage of total disk storage. The Ram Disk is always full, because it expands and contracts dynamically as you add and delete files.

Errs: The Errs referred to here are soft errors. AmigaDOS uses a complex system of keys and checksums to ensure the integrity of the file system and, occasionally, the checksums don't match. A soft error can be serious, and you should try to isolate and eliminate them. For more details, see the DiskDoctor command in the next chapter.

Status: A disk can be in one of two states — read/write or read Only. Obviously, you can't write to a read only disk. You set the status of a hard disk, hard-disk partition, or floppy disk using the Lock command (Chapter 9). You can also set the status of a floppy disk with its write-protect tab.

Name: The Name of a disk is its volume name — the name it goes by under the Workbench interface. Using AmigaDOS, you can use either the volume name or the device name.

The bottom part of the Info display lists the volumes currently available to the file system and labels those that are mounted. An unmounted volume will be

available if it contains a directory that is part of the search path or is assigned a logical directory.

The Info command takes one optional argument, DEVICE. It lets you specify a device on which you want information. For example, to see the information on your internal floppy drive, you enter:

```
INFO DF0:
```

The DEVICE argument is rarely used because you get the same information — and more — by simply entering Info.

Date and SetClock: Timely Matters

The Amiga has two clock/calendars, the system clock and the battery-backed one, so AmigaDOS supplies two commands — Date and SetClock — that let you work with both. (The SetDate command, covered in the next chapter, is concerned with the time/date stamps on files, not with the clocks.)

The original Amiga, the A1000, has only the system clock, which the Clock program and AmigaDOS use to put time/date stamps on files. Whenever you turn on an Amiga 1000, you have to set the system clock using the Date command to have accurate time/date stamps and an accurate clock. With the Amiga 2000 and models that followed, Commodore built in a battery-backed clock/calendar (or supplied it as an option with the A501 memory board for the A500) that you only needed to set once. The clock keeps running after you power down. The problem was that AmigaDOS, the Clock, and any other programs that need time and date information were written to use the system clock, not the battery-backed one.

The solution to the two-clock dilemma is the SetClock command. SetClock doesn't let you directly set the battery-backed clock; instead, SetClock lets you set the battery-backed clock from the system clock and then set the system clock from the battery-backed clock. Let's see how Date and SetClock work together in practice.

To get all the clocks in your system running properly, you first use the Date command to set the system clock. Date has the following template:

```
DAY,DATE,TIME,TO=VER/K
```

When used without arguments, Date returns the current data and time settings of the system clock. The TO argument lets you direct the output of the command to a file or device, such as your printer. The DATE and TIME arguments let you set the data and time, respectively.

The DATE argument takes the form DD-MMM-YY, where DD is the day of the month (leading zeros are not required), MMM is the first three letters in the month name, and YY is the last two digits in the year. Note that you can't set the clock to a date before January 1, 1978. Years from 00 to 77 are considered 2000 to 2077. Commodore is ready for the 21st century.

The DAY argument lets you substitute a day of the week for the DATE argument. When you use a day of the week, the command substitutes the date the day will fall upon in the next week. Of course, it makes this substitution based upon the current setting of the clock; if the clock is incorrect, the date you get entering a day of the week will also be incorrect.

The TIME argument takes the form HH:MM:SS, where HH is the hour of the day (0-23), MM is the minute (0-59), and SS is the second (0-59). The seconds part of the time is optional. Thus, to set the time and date to 6:55 PM, July 5, 1991, you enter:

```
DATE 18:55 5-jul-91
```

Once you have the system clock set, you use it — and the SetClock command — to set the battery-backed clock.

SetClock has the following template:

```
LOAD/S,SAVE/S,RESET/S
```

The SAVE/S argument tells SetClock to set the battery-backed clock with the time and date in the system clock. If you've previously set the system clock with the Date command, this argument will ensure that the battery-backed clock has the correct time and date.

Having set the battery-backed clock once with

```
SETCLOCK SAVE
```

you need never set it again unless you move to another time zone or wish to keep up with the daylight-savings changes. You still need to set the system clock, however, every time you turn on your system. Instead of using Date to enter the time and date manually, you use the LOAD/S option of SetClock.

The LOAD/S option is the opposite of SAVE/S. It sets the system clock from the battery-backed clock. If you do this once every time you turn on your system, you'll always have the correct time and date in the system clock, and thus always have the correct time and date for the Clock program and for AmigaDOS time/date stamps. When you examine the standard AmigaDOS Startup-sequence in Chapter 10, you'll see that the first command is:

```
SETCLOCK LOAD
```

The last SetClock argument, RESET/S, is rarely used. If you find that some rogue program has messed up your battery-backed clock and you're unable to set it properly with the SAVE/S option, try this argument before attempting another save.

In conclusion, to get all your clocks synchronized and running correctly, you first set the time and date with the Date command, save that information to the battery-backed clock with the SetClock SAVE option, and then, every time you boot your computer, execute a SetClock LOAD command, preferably from your Startup-sequence.

Avail: Available Memory

Memory is probably the most precious resource on you Amiga. The amount you have determines how many programs you can run at once and even how fast the programs execute. The Avail command tells you how much memory you have on your system and how much of that has yet to be allocated to program.

Avail's command template is:

CHIP/S,FAST/S,TOTAL/S,FLUSH

When you enter Avail without arguments, it reports on the amounts of chip and fast RAM you have on your system as well as on the total of the two. Chip RAM is memory used to store graphics and sounds, fast RAM is used primarily to hold executing programs. If your computer has only chip RAM, it will use this memory to store both its graphics data and its programs. An Amiga can't operate without chip RAM.

The Avail command reports four pieces of information about the chip, fast, and total memory on your system (see Figure 8-11). It reports on the available memory (that which has yet to be allocated to any program), the memory in use, the maximum memory (the combination of the available and in-use memory), and the largest block of memory available.

The largest block is very important, because the Amiga OS allocates memory in contiguous blocks. As programs open and close, they fragment memory into discontinuous blocks. If you want to run a program that requires 300K of memory and the largest block available is 250K, you won't be able to run the program, even if the total available memory is 500K. In such a case, you can try the FLUSH/S argument, which forces Amiga OS to flush its memory lists of any de-allocated memory that hasn't been returned to the available memory

pool. If this doesn't get you a contiguous block large enough for your program, you may have to close down any executing programs and reboot your system. After a reboot, memory is much less fragmented than it is after the machine has been in use for a while.

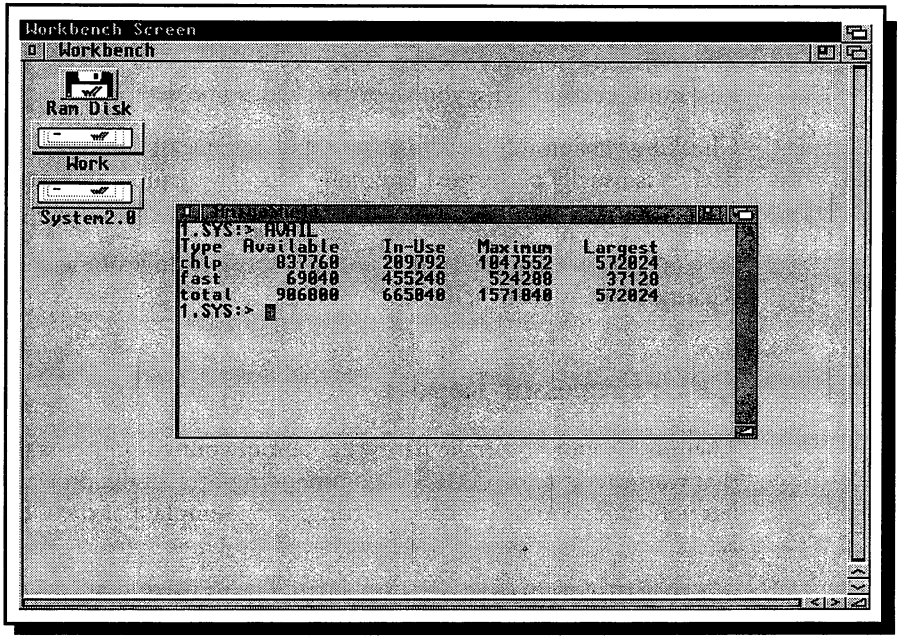


Figure 8-11 Memory Information

What Info does for disks, Avail does for memory. The Total row is simply the sum of the chip and fast memory.

The CHIP/S, FAST/S, and TOTAL/S arguments let you selectively examine chip, fast, and total memory, respectively. These options are rarely used because the default, showing all three, is easier to access and more informative.

The Version Command

Having the correct version of the system software is very important. Some programs won't run if you don't have a specific version of Kickstart or a system library. The Version command lets you check the version number of Kickstart, Workbench, and the system libraries and devices. It has the template:

NAME,VERSION/N,REVISION/N,UNIT/N

NAME is the name of the library or device you want to check. If you don't use a name, the command returns the version numbers of your Kickstart and Workbench. The VERSION/N and REVISION/N arguments let you compare the numbers the command returns with numbers you provide. If the command finds numbers that are greater than or equal to the version and revision numbers you provide on the command line, it sets a return code of 0. Otherwise, the return code is 5. The UNIT/N argument lets you specify the unit number of a device you want checked; some devices have multiple units.

Checking version numbers to ensure the presence of compatible libraries and devices is usually performed internally by software programs. It is not something you have to know about unless you are a software developer.

As you'll see in Chapter 10, Version also sets two environment variables, Kickstart and Workbench.

CPU: Processor Report

The most important single hardware item in your computer is the CPU chip that runs the system. The CPU command reports on the processor itself and lets you change some processor settings. The template of the CPU command is:

```
CACHE/S,BURST/S,NOCACHE/S,NOBURST/S,DATACACHE/S,
DATABURST/S,NODATACACHE/S,NODATABURST/S,
INSTCACHE/S,INSTBURST/S,NOINSTCACHE/S,
NOINSTBURST/S,FASTROM/S,NOFASTROM/S,TRAP/S,
NOTRAP/S,NOMMUTEST/S,CHECK/K:
```

If you simply enter:

```
CPU
```

The command returns information about your CPU. For example, on my Amiga 3000, the command responds with:

```
68030 68882 FastROM (INST: Cache Burst) (DATA: Cache
NoBurst)
```

This tells me that my computer has a 68030 processor, a 68882 math coprocessor; that the Kickstart ROM has been moved to fast RAM, and that the instruction cache is active and enabled for burst mode while the data cache is active but not enabled for burst mode.

A detailed description of all the options of the CPU command would require a lengthy dissertation on microprocessor design that is beyond the scope of this book. From the names of the switches, you can deduce that the CPU command lets you enable and disable your CPU's instruction caches (INSTCACHE/S, NOINSTCACHE/S), data caches (DATACACHE/S, NODATACACHE/S), or both (CACHE/S, NOCACHE/S). You can also enable or disable burst mode on the instruction (INSTBURST/S, NOINSTBURST/S) and data (DATABURST/S, NODATABURST/S) pipelines, or on both at once (BURST/S, NOBURST/S). Of course, to use burst mode you must have memory chips that are capable of burst mode. You can enable and disable exception trapping with TRAP/S and NOTRAP/S, respectively, and test for the presence or absence of an MMU (memory-management unit) with NOMMUTEST/S.

The FASTROM/S argument lets Amigas equipped with an MMU move Kickstart from ROM to fast RAM, where it can execute library calls much faster. (The 68030 has an MMU built-in; with a 68020, you need a separate MMU chip. The 68000 does not support an MMU.) Once Kickstart 2.0 is in ROM, Amiga 3000's will not longer need the FASTROM/S option because the A3000's ROMS are 32 bits wide. NOFASTROM/S turns off this feature. CHECK/S, when followed by CPU, MMU, or FPU, performs a check of these hardware components.

What are you going to do with the CPU command? Hopefully, nothing. If your computer is running fine, don't mess with these options. If you buy a 68020 or 68030 accelerator for your Amiga 500 or 2000 and you have extra fast RAM, you may want to enable the FASTROM/S option. Otherwise, you should keep the CPU settings to the factory defaults.

Conclusion

Using the commands detailed in this chapter, you can gather information about the internal organization of AmigaDOS and change that organization to suit your needs. Although commands like CPU and Status are a bit esoteric for the average user, you need a good working knowledge of most of these commands to get the most out of your system. List and Assign are particularly important.

The next chapter deals with commands that let you change the actual files and devices on your system. This is where you can put the information you gleaned from the commands in this chapter to work.

– 9 –
Manipulating Files and Devices

In Chapter 8, you learned about the many commands that let you investigate, and sometimes change, the state of the AmigaDOS file system, devices, and processes. The commands in this chapter are different; they let you physically modify different aspects of AmigaDOS. They let you copy files, edit text, mount devices, change protection flags, and more. They are more concerned with *what* AmigaDOS does, rather than how it does it.

Working with the File System

Like the commands in the last chapter, those discussed here work with both files and devices. Because you spend most of your time with AmigaDOS working with files, I'll start there. The commands that let you create and modify files, directories, and volumes are Copy, Delete, DiskDoctor, FileNote, Install, Join, Lock, MakeDir, MakeLink, Protect, Relabel, Rename, SetDate, and Sort.

Copy: Copying Files

Along with Dir and List, Copy is one of the most frequently used AmigaDOS commands. It lets you duplicate individual files or whole sections of the AmigaDOS file structure. The template of the Copy command is:

```
FROM/A/M,TO/A,ALL/S,QUIET/S,BUF=BUFFER/K/N,
CLONE/S,DATES/S,NOPRO/S,COM/S,NOREQ/S
```

The FROM/A/M argument stands for the source file or files you wish to copy. You must always include this argument on the command line; you can include

many sources if you wish. The TO argument gives the destination file or directory for the copy source. In its simplest form, the command looks like this:

```
COPY file1 TO file2
```

This command copies the contents of file1 to file2. If file2 does not exist before you execute Copy, the command creates it. If it does exist, the command overwrites the old contents with the contents of file1. The system doesn't warn you, so be careful or you'll overwrite an important file.

In addition to simple filenames, the FROM and TO arguments can also be pathnames. For example,

```
COPY file1 TO RAM:file2
```

copies file1 from the current directory to a file named file2 on the RAM: disk. You can also use logical names in the FROM and TO arguments:

```
COPY Extras2.0/Devs/Printers/EpsonQ TO DEVS:Printers/EpsonQ
```

copies the EpsonQ printer driver from the Extras2.0 disk to the Printers directory of the DEVS: directory.

When you copy a file outside of its directory, you can use the original filename for the destination file by not supplying a filename in the TO pathname. In the above example, you get the same results by entering:

```
COPY Extras2.0/Devs/Printers/EpsonQ TO Devs:Printers
```

This command copies the EpsonQ driver into the destination directory and gives the destination file the same name as the source file. Note also that the TO keyword is optional; as long as you put the FROM argument first and the TO argument second, you can omit the keywords. (I always use the TO keyword, however, because it's a good visual clue for what's happening.)

If you specify a directory for the FROM argument, Copy copies all the files (not the directories) inside the FROM directory. In this case, the TO argument *must* specify a directory. For example,

```
COPY System2.0/System TO System2.0/Utilities
```

copies all the files in the System directory of a hard-drive-equipped Amiga to the Utilities directory. However,

```
COPY System2.0/System TO Utilities/Clock
```

produces the following error message:

```
Destination must be a directory
```

You can also use wildcard patterns for the FROM argument. For example,

```
COPY SYS:System/#?m#? TO RAM:
```

copies all the files in the System directory that have the letter m in their names to the Ram Disk. Note that when you copy more than one file at a time, either by putting multiple source files on the command line, specifying a directory as the source, or a wildcard pattern, the Copy command reports all the files it has copied. For example, if you executed the above command, Copy would output the list shown in Figure 9-1.

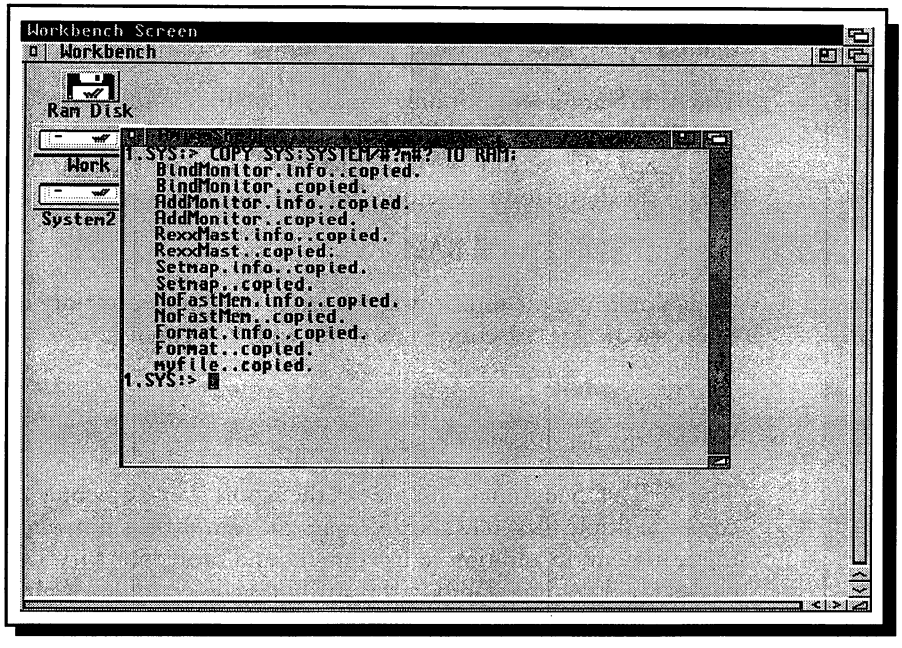


Figure 9-1 Copying Multiple Files

When you copy multiple files with a wildcard, Copy reports every file that it copies. Note that you must specify a directory as the destination of a multiple-file copy.

An important point about the Copy command is that multiple-file copies always go to a directory. If you specify an existing file as the destination of the multiple-file copy, the system sends an error message. If you specify a filename that does not exist, Copy creates a directory with that name and copies the files

indicated into it. For example, if you try to use a wildcard as the destination, Copy has a surprise for you. For example, you might think that

```
COPY SYS:System/#?m#? TO RAM:Mfile_#?
```

would copy all the files from System whose names contain the letter m to the Ram Disk and append "Mfile_" to each of their names. What happens, however, is that Copy creates a directory named Mfile_#? on the Ram Disk and then copies the files into it. The TO argument does not support wildcard characters.

While the Copy command does copy the contents of a file exactly, it does not copy everything associated with the file; specifically, it does not copy the time/date stamp or any comments. It gives the destination files a time/date stamp that reflects the time of their creation by the Copy command. One attribute that is copied, however, is the protection status of the source file.

The ALL/S argument not only copies all the files in the source directory, but also all the directories and files below the source in the AmigaDOS file structure. Copy will recreate the file structure of the source directory as it copies files to the destination directory. For example, if you enter:

```
COPY FONTS: TO RAM: ALL
```

Copy duplicates the directory structure of the Fonts: directory in the RAM: directory. If, after executing this command, you enter:

```
DIR RAM: ALL
```

You get the list shown in Figure 9-2.

The QUIET/S argument suppresses the report that Copy outputs when it copies multiple files. CLONE/S copies the source file's time/date stamp and the comment, in addition to its contents and protection bits. DATE/S includes the date-time stamp in the attributes copied to the destination, while COM/S includes the comment. Thus, the following two commands are equivalent:

```
COPY file1 TO file2 CLONE
COPY file1 TO file2 DATE COM
```

The NOPRO/S argument tells Copy not to duplicate the protection bits of the original to the destination file. Instead, Copy sets the r, w, e, and d flags on the destination file, regardless of the protection settings of the source file. The NOREQ/S argument suppresses any requesters generated by the command. For example, if you try to use a volume that is not mounted as the source or

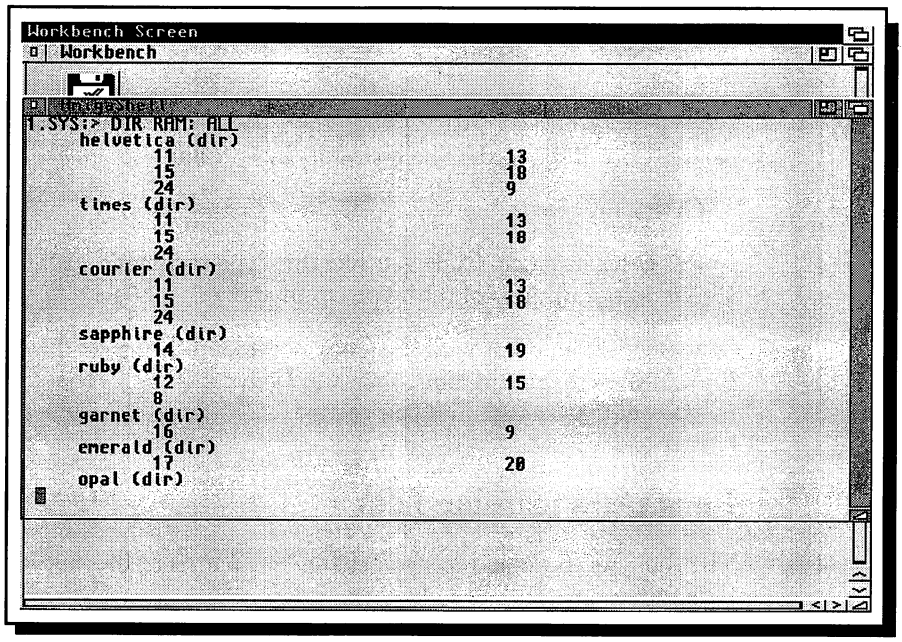


Figure 9-2 Copy ALL

When you use the ALL option, the Copy command duplicates the entire AmigaDOS file structure at and below the point you specify as the source of the copy operation. In the example, one Copy command copied all the files and directories in the Fonts: directory to the RAM: disk.

destination of a copy, the command will normally put up a requester asking you to insert the unmounted volume. If you specify NOREQ on the command line, however, Copy will instead output the message:

```
copy: device (or volume) is not mounted
```

and set a return code of 20. NOREQ/S is used mostly in command scripts where you can test for the return code and take action if the command fails.

Besides duplicating existing files, Copy also lets you create files from the keyboard. If you enter:

```
COPY * TO my_file
```

the command copies whatever you type into the Shell window to a file named my_file, which Copy creates if it does not already exist. To close the file and bring back the prompt, enter CTRL-\.

Copy lets you copy files to devices as well as to files. You can copy files to the printer device by entering:

```
COPY my_file TO PRT:
```

or even copy a file to the Shell window by entering:

```
COPY my_file to *
```

In this latter instance, Copy acts like the Type command. (Of course, it lacks all the output options of the Type command.)

When you experiment with the Copy command, I suggest that you use the Ram Disk as the destination. That way, you can simply reboot your computer to erase all the practice files you created.

Rename: Renaming Files

The Rename command lets you change the names of files and directories. In its simplest form, it looks like this:

```
RENAME harry george
```

This command will change the name of the file harry to george as long as a file named george does not already exist in the same directory. If george already exists, the system sends the following message:

```
Can't rename harry as george because object already exists.
```

Beyond renaming a single file within the same directory, the Rename command resembles the Copy command. In fact, once you start renaming multiple files and directories, you can think of Rename as a copy operation that automatically deletes the source files and directories.

The template for Rename is:

```
FROM/A/M,TO=AS/A,QUIET/S
```

Like Copy, the Rename FROM/A/M argument can be a single file, multiple files, a file pattern, or a directory. Again, as with the Copy command, if you specify more than one file in the FROM/A/M argument, you must specify a directory as the TO=AS/A argument. If you specify a file as the destination of a multiple-file rename or specify a name that doesn't exit you get an error message. Unlike Copy, however, Rename doesn't create directories. The QUIET/S argument suppresses the messages that Rename produces as it renames multiple files.

Rename could just as easily be called Move, because it lets you move files and directories around in the AmigaDOS file structure. For example, assume you have a data disk named Data: that contains three directories — Letters, Reports, and Memos — in its root directory, and multiple files in each subdirectory. (Figure 9-3 shows what the disk looks like when you use the ALL option of the Dir command.) Now, suppose that you decided that memos was simply a subcategory of letters, so you wanted to move the Memos directory inside the Letters directory. You'd enter:

```
RENAME Data:MEMOS TO Data:LETTERS
```

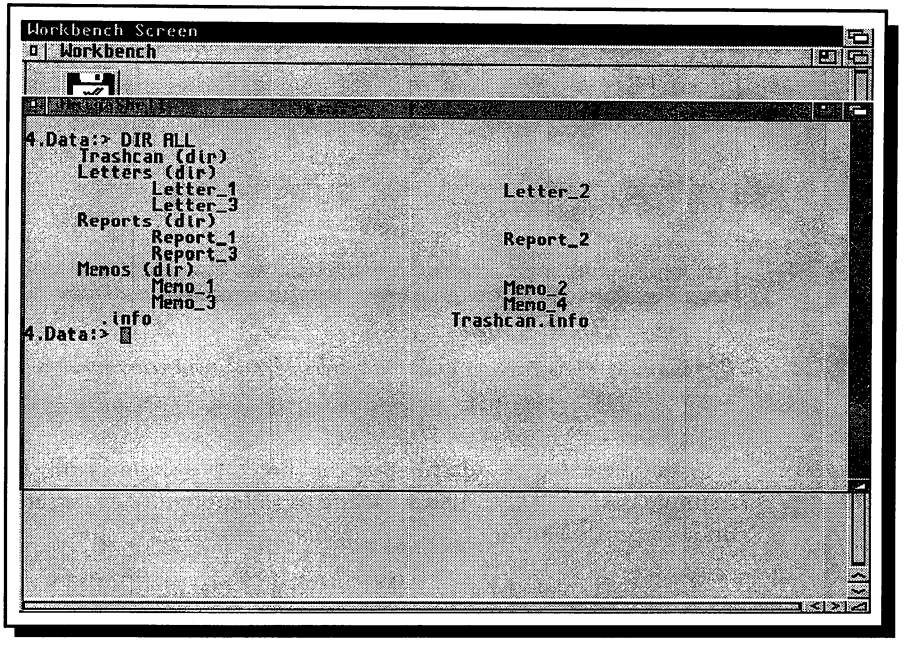


Figure 9-3 Structure of Data:
This listing shows how the files and directories are arranged on the Data: disk before any Rename operation.

Rename responds:

```
Renaming Data:Memos as Data:Letters/Memos
```

Figure 9-4 shows the new directory structure of the disk.

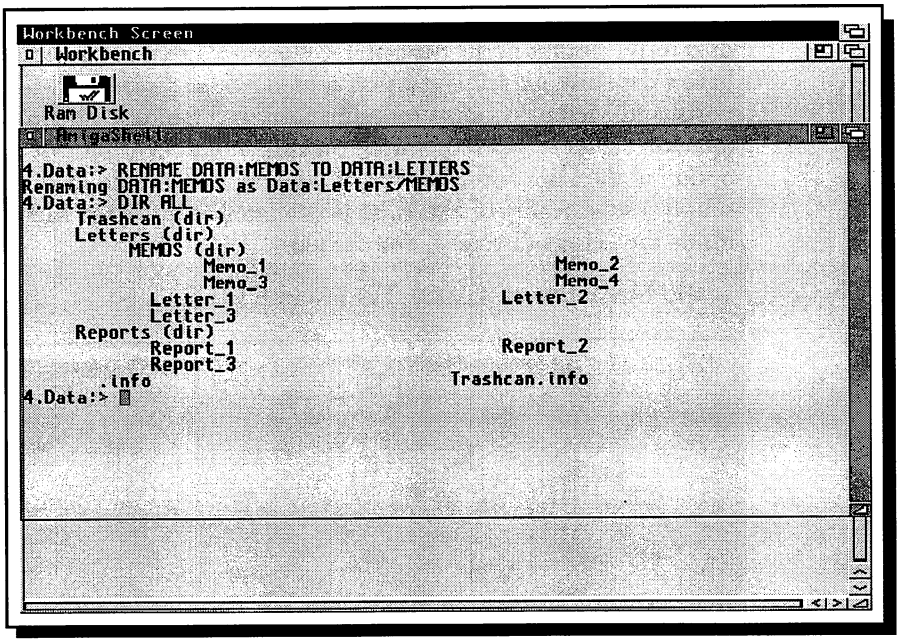


Figure 9-4 Renaming Memos

By renaming Data:Memos to Data:Letters/Memos, you rearrange the directory structure of the Data: disk.

Now suppose that you decide that the directories should really be structured as a simple hierarchy with Letters at the top, Reports under Letters, and Memos under reports. You can accomplish this by entering:

```
RENAME Data:LETTER/MEMOS TO Data:REPORTS
RENAME Data:REPORTS TO Data:LETTERS
```

Figure 9-5 shows the results of your directory restructuring. Note that if you substitute the name Move for Rename in the commands, you get a much clearer idea of what's actually going on. In fact, in my Startup-sequence, I've added the command:

```
ALIAS Rename Move
```

I use Rename when I want to rename a file and Move when I want to re-arrange my directory structure. The two names load and execute the same command, but better reflect the different functions of the Rename command.

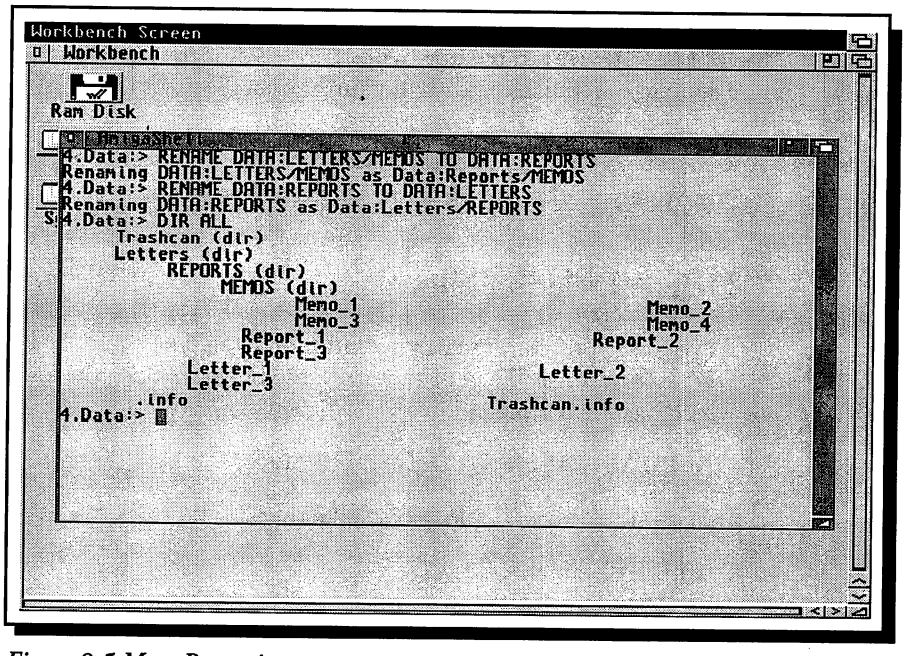


Figure 9-5 More Renaming

This figure shows the results of two more Rename commands. Except when used on files in a single directory, Rename works like a Move command.

A final note about Rename: It does not work across disks and devices. Any new name you give a file or directory must be on the same disk as the old name. Thus, if you enter:

RENAME SYS:System TO RAM:

an error message stating that you can't rename across disks and devices appears.

Relabel: Renaming a Volume

AmigaDOS provides a special command — Relabel — for renaming disk volumes. Its template is:

DRIVE/A,NAME/A

The DRIVE/A argument is the device name of the drive that contains the volume you want to rename, while NAME/A is the new volume name. For example, to rename the disk in your internal floppy to Data:, you enter:

```
RELABEL DF0: Data
```

Note that you don't put a colon after the new volume name.

Delete: Deleting Files

As you use your computer more and more, you'll accumulate a lot of files that you no longer use. To get rid of unwanted files, you use the Delete command, which has the template:

```
FILE/M/A,ALL/S,QUIET/S,FORCE/S
```

In its simplest form, Delete looks like this:

```
DELETE my_file
```

The above command deletes the file named my_file in the current directory. From the template, the FILE/M/A argument lists the file or files that you want to delete. You can list a single file, multiple files, a file pattern, or a directory. For example, to clear a directory named Letters of all files whose names include the letter x, you enter:

```
DELETE LETTERS/#?x#?
```

To delete a directory, it must be empty. For example, if you have a directory named Pictures on your data disk that contains many files and subdirectories and you enter:

```
DELETE Data:PICTURES
```

The system tells you:

```
Data:PICTURES Not Deleted: directory not empty
```

You now have two options: You can go into Pictures and all the directories below it and delete all the files and directories individually or you can enter:

```
DELETE Data:PICTURES ALL
```

The ALL/S argument deletes a directory and all the files and directories below it in the file structure. The QUIET/S argument suppresses the messages that Delete produces when disposing of multiple files.

Normally, if a file or directory does not have its delete bit set, you can't delete it. Using the FORCE/S argument, however, you can delete such files.

Delete is a very powerful command that, unlike the Delete item in the Icons menu, does not produce any requesters that give you a chance to change your mind. Be very careful using wildcards and the FORCE/S and ALL/S options with Delete; you could inadvertently delete some vital information. If you find that you've started a disastrous multiple-file delete, hit CTRL-C to abort the command and minimize the damage.

Setting Protection Bits

Each Amiga file and directory has a set of protection bits that give information about the file and indicate the actions you can take with it. The eight defined protection bits are Archived (a), Executable (e), Hidden (h), Pure (p), Read (r), Script (s), and Write (w).

Archived: The archived bit is used primarily by disk-backup utilities, which will set the archived bit when they back up a file. If, sometime later, the file is altered by an editor, renamed, or copied to, the archive bit is removed. When the backup utility is next run, it knows that it must backup this file. If the archive bit is still set, indicating that the file has not been altered, the back up utility knows that it doesn't have to archive this file again.

Deletable: With this bit set, you can delete the file; if it is absent, you can't delete the file unless you use the FORCE/S option of the Delete command. This bit works as advertised.

Executable: If an executable file has this bit set, then you can execute the command. If an executable file does not have the Executable bit set, you can execute the file. If a nonexecutable file has this bit set, you still can't execute the file. In other words, the Executable bit identifies executable files, it doesn't make them executable.

Hidden: When set, the hidden bit is supposed to keep a file from being listed with List or Dir. Unfortunately, this feature has never been implemented. For now, the Hidden bit is meaningless.

Pure: The Pure bit identifies program files as reentrant and reexecutable. The Resident command checks this bit when it tries to make a command resident. Note that setting the bit does not make a program pure; it merely makes the program palatable to the Resident command. Don't change the factory setting of the Pure bit on any file that comes with AmigaDOS 2.0; making a command resident that isn't really pure is not a good idea.

Readable: The Readable bit is supposed to identify those files that any editor or the Type command can read. When absent, the file should be unreadable. In practice, the AmigaDOS commands ignore the setting of the Readable bit, although some commercial programs pay attention to it.

Script: The Script bit identifies AmigaDOS script files. With the bit set, you can execute a script by simply entering its name on the command line, without having to call it as an argument to the Execute command (see Chapter 10). Scripts called by Execute don't need this bit set.

Writable: With the Writable bit set, you can write to a file; if it is absent, you should be unable to write to the file. In practice, the AmigaDOS commands ignore this bit, although some commercial programs do not.

You set and clear the protection bits using the **Protect** command. It has the template:

```
FILE/A,FLAGS,ADD/S,SUB/S,ALL/S,QUIET/S
```

The FILE/A argument lists the name of the file or directory whose protection bits you wish to change. Note that the protection status of a directory is completely independent of the protections status of the files it contains. FILE/A can also be a file pattern.

The FLAGS argument identifies the protection bits you want to set or clear. You must use the abbreviation of the name of the bit, which is the first letter of the name. ADD/S and SUB/S indicate whether you are adding (setting) the indicated bit or subtracting (erasing) it. For example, to make your Startup-sequence file nondeletable, you enter:

```
PROTECT S:Startup-sequence d SUB
```

You can set more than one flag at once. For example,

```
PROTECT my_file shapdwre ADD
```

sets all the bits on the file my_file. The bits don't have to be in any particular order.

The ADD/S and SUB/S keywords are mutually exclusive; they can't be on the same command line. You can substitute a plus sign (+) for ADD, and a minus sign (-) for SUB. Note that the plus or minus sign can come before or after the flags, but, unlike the ADD and SUB keywords, there must be no space between the sign and the flags. Thus, to make your Startup-sequence file deletable again, you can enter:

```
PROTECT S:Startup-sequence +d
```

If you use a directory name with the ALL/S option, Protect adds or subtracts the bits you indicate from every file and directory in the directory, and from every file and directory below it in the file structure. For example,

```
PROTECT SYS:PREFS -d ALL
```

protects every file in or below your Prefs directory from deletion. The QUIET/S option suppresses the messages that Protect outputs when it works with multiple files at a time.

Lock: Write Protecting a Volume

The Lock command lets you write protect a floppy disk, hard disk, or hard-disk partition. (You can't copy or save anything to a write-protected disk.) Lock's template is:

```
DRIVE/A,ON/S,OFF/S,PASSKEY
```

Without an argument, Lock returns the write-protect status of a disk. For example:

```
LOCK SYS:
```

tells you whether or not your system disk is write-protected. The keywords ON/S and OFF/S let you turn write protection on and off, respectively. Thus, to write-protect your system disk, you enter:

```
LOCK SYS: ON
```

The PASSKEY is a four-character string that you can specify when you lock a disk or partition. When you use it with the ON/S argument, you won't be able to unlock the disk unless you enter the same string with the OFF/S argument.

Keep in mind that if the write-protect tab on a floppy disk is set to protect, you can't unlock the drive using the Lock command, although you can write protect such a disk if its write-protect tab is set to enable.

SetDate: Setting a Time/Date Stamp

Whenever you create or modify a file, AmigaDOS automatically stamps the file with the time and date of creation or modification. (AmigaDOS doesn't actually *stamp* anything; it saves the time and date with the file). The SetDate command lets you manually input a time/date stamp, or have AmigaDOS create a new one. The template for the SetDate command is:

```
FILE/A,DATE,TIME,ALL/S
```

The only argument required for the SetDate command is FILE/A, which lets you specify the file or directory to which you want to give a time/date stamp. The argument can be a pattern, so you can change more than one file at a time. The DATE and TIME arguments take the forms DD-MMM-YY and HH:MM:SS, respectively. (See the Date command for an explanation of these formats.)

If you enter the command and the FILE/A argument without a DATE or TIME, SetDate will stamp the file with the current contents of the system clock. If you use the ALL/S argument, SetDate will stamp all the files that match the FILE/A argument in the current directory and every directory below the current one. Thus, to give every file and directory on a disk in the internal drive a time/date stamp that reflects the current setting of the internal clock, you enter:

```
SETDATE DF0:#? ALL
```

FileNote: Commenting on Files

When using AmigaDOS, you should try to make your filenames as descriptive as possible. Descriptive filenames, however, tend to be long and annoying to type repeatedly. AmigaDOS provides relief from this dilemma by letting you append a 79-character comment to a file using the FileNote command. This comment is printed when you use the List command to view the files in a directory.

The template for FileNote is:

```
FILE/A,COMMENT,ALL/S,QUIET/S
```

The FILE/A argument lets you specify the file or directory to which you want to append a comment. The argument can be a name or a pattern; you can thus give multiple files the same comment.

The COMMENT argument is the string of characters that make up the comment. As with just about everything else in AmigaDOS, you have to put quotes around the entire comment if it contains a space. If you want to include quotes inside the comment, you have to precede each with an asterisk *and* put quotes around the entire comment, even if it doesn't contain a space. For example, to append the comment Outline for "Operation Recycle" to a file named recycle.txt, you enter:

```
FILENOTE recycle.txt "Outline for *"Operation Recycle.*""
```

Note that the COMMENT argument is not required. When you leave it off, you delete the comment from the specified file or files. FileNote displays a message with the filename and the word "Done" whenever it creates a new comment. The QUIET/S option suppresses this message.

The ALL/S option appends a comment to every file and directory in and below the current directory that matches the FILE/A argument.

MakeDir: Creating Directories

The most important reason for having a hierarchical file structure such as the one used by AmigaDOS is that it lets you organize your files in a logical manner. You can group like files, and separate dissimilar ones. One of the most important commands in helping you achieve this goal is MakeDir.

MakeDir is short for make directory; its purpose is to create a place where you can store files that share a similar function. The template of the MakeDir command is:

```
NAME/M
```

The lone argument is simply the name of a directory you want to create. For example, to create a directory named Disk.Utils in the Utilities directory of your system disk, you enter:

```
MAKEDIR SYS:UTILITIES/Disk.Utils
```

If you were already in the Utilities directory, you wouldn't need the absolute pathname for the new directory. You could simply enter:

```
MAKEDIR Disk.Utils
```

The /M modifier means you can create multiple directories with one MakeDir command by separating the names on the command line with spaces. Once you've created a directory, you can use it to store other files and directories. Giving your directories logical names and a logical organization makes working with your computer a lot easier.

Join: Concatenating Files

The Join command lets you create a file by copying the contents of multiple files into the destination file. It has the following template:

```
FILE/M,AS=TO/K/A
```

The FILE/M argument consists of any number of files or file patterns that you wish to combine into one large file. None of the input files are changed by joining them. The AS=TO/K/A argument is the name of the combined file. It can't be the name of one of the input files. For example, to combine all the script files in your S: directory into one file named all_scripts, you enter:

```
JOIN S:#? AS all_scripts
```

You can combine all types of files — programs, data files, icon files, and so on — with Join, but most of the time you will use Join to combine text files. For example, if you've used Dir with the ALL option to create files that list the contents of every disk you own, you can create a master file of the contents of all your disks by using the Join command.

Sort: Sorting Text Files

Let's say you captured a list of all the files in your Sys:System directory by entering:

```
LIST >files.list Sys:System NOHEAD
```

When you type the file, you get the output shown in Figure 9-6, accurate but unorganized. To sort the filenames in alphabetical order, you enter:

```
SORT files.list TO sorted.list
```

When you type sorted.list, you'll see that its contents are alphabetized (see Figure 9-7) thanks to the Sort command, which sorts the lines of a text file. Its template is:

```
FROM/A,TO/A,COLSTART/K,CASE/S,NUMERIC/S
```

The FROM/A argument names the file you want to sort, while TO/A identifies the sorted file produced by the command. Note that the two files can be the same.

The COLSTART/K argument specifies the first character of the key to the sort. A sort key is the character or group of characters in each line that the Sort command tests to determine where each line should appear in the sorted output. If necessary, the Sort command will use the entire line as a sort key. By default, the key to a sort begins with the first character on each line. You can change the beginning character of the key with the COLSTART/K argument. For example, if you sort the files.list with a different COLSTART/K, you get different results. Enter the following:

```
SORT files.list TO newsort.list COLSTART 2
```

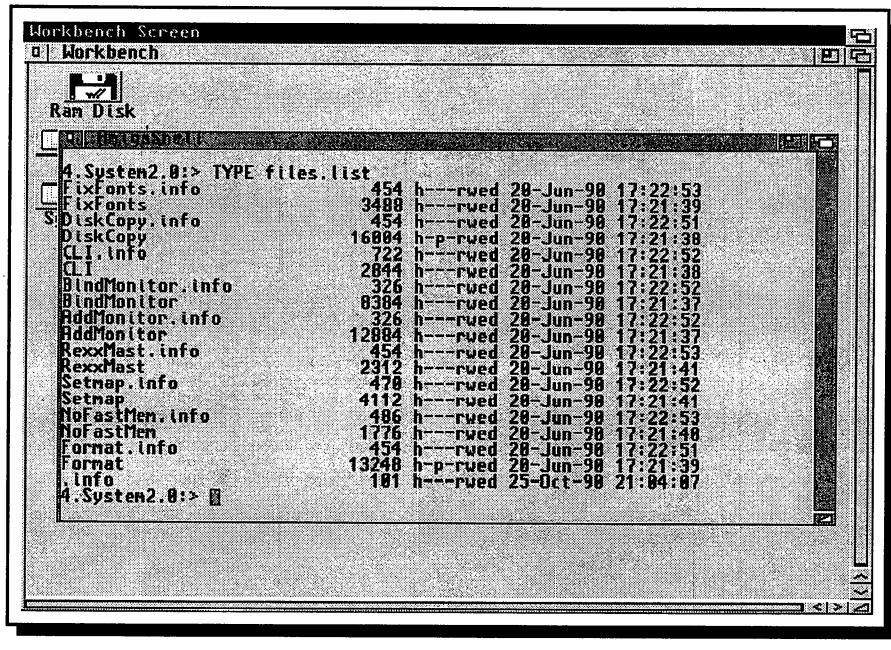


Figure 9-6 Presorted List

The List command gives more information than Dir, but it doesn't alphabetize its output. Here is a file created by List before being sorted.

When you type the file newsort.list (see Figure 9-8), you'll see that the lines are sorted based upon the alphabetical position of the *second* character in each line, instead of the first.

Normally, the Sort command is case-sensitive, ranking uppercase letters before lowercase ones ("Z" comes before "a"). If you use the CASE/S argument, Sort does not differentiate between upper- and lowercase letters ("a" comes before "Z"). Note that the Amiga OS 2.0 documentation from Commodore explains this situation backwards.

The NUMERIC/S switch lets you sort numbers based upon their numeric values, instead of their character values. For example, enter the following lines into the Shell:

```
COPY * TO number.list
12345
9
3333
3
```

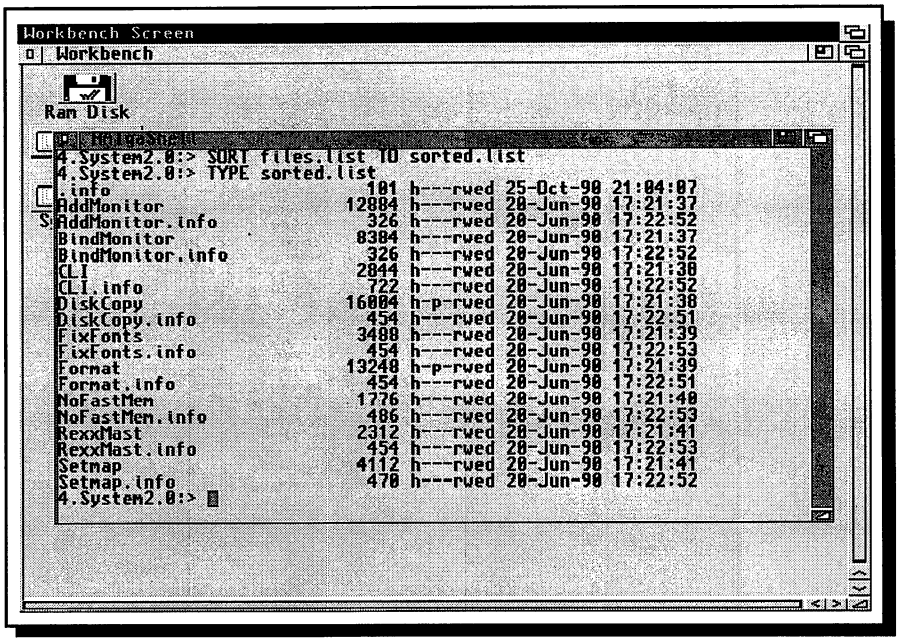


Figure 9-7 Sorted List

Using the sort command on the file shown in Figure 9-6, you produce a file sorted by the first letter of the filenames.

```
123
98
1
0
344
100
CTRL-\
```

(Be sure you press the CTRL key and the Backslash key together for the last line; do *not* enter the characters C, T, R, L, -, and You just created a file named number.list in the current directory that contains the lines of data you typed. Now, sort the file with the following command:

```
SORT number.list TO sort1.list
```

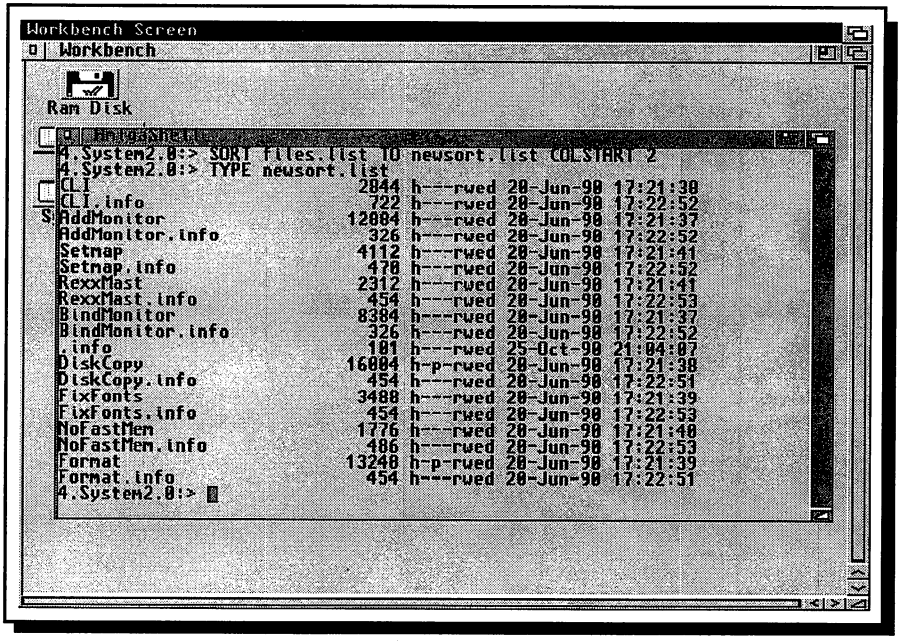


Figure 9-8 An Alternate Sort Key

Using the COLSTART argument, you can sort on some value other than the first letter of the filenames. This is the same file shown in Figures 9-6 and 9-7, but sorted on the second character of the filename.

and type the output file. Figure 9-9 shows the output list. Note that its order is not based on the numbers' numeric values, but on their character values. To sort the file on purely numeric considerations, you enter:

SORT number.list TO sort2.list NUMERIC

Now, when you type the output file (see Figure 9-10), you'll see the lines of the file arranged by their numeric values.

Commodore designed the Sort command to work with text files, but you can try to sort binary files. Because Sort delineates lines by carriage return characters — which binary files lack — such sorts are not very successful or meaningful.

Before tackling an extremely large file, consider that the Sort command can eat up stack space very quickly. You should increase the size of your stack with the Stack command. When I sort files, I normally use a stack of 50,000 bytes. I reduce it again to 4,096 when I'm finished.

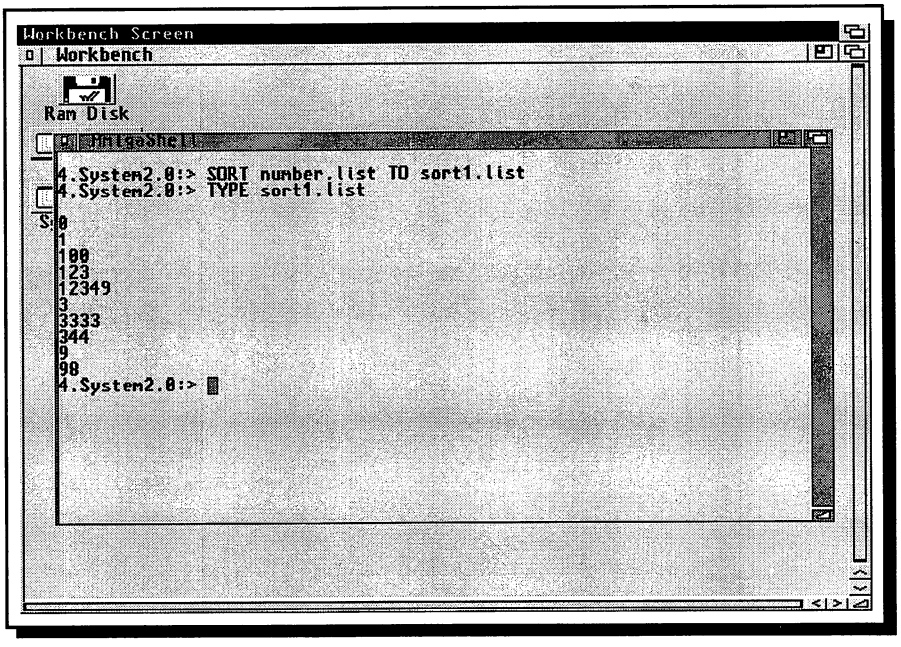


Figure 9-9 Sorting Numbers

The Sort command treats numbers like any other character, and sorts them by their character values, instead of their numeric values.

MakeLink: Creating Links Between Files

The MakeLink command creates a file that points to another, already existing file. Whenever you use the name of the first file in a command, the command actually accesses the second file. For example, if you enter:

```
MAKELINK SYS:link TO C:Dir
link
```

you get a directory listing of the current directory! The link file is actually a copy of the file to which it is linked. If the linked file changes (C:Dir here), however, the link file (Sys:link) automatically changes also. The template of MakeLink is:

```
FROM/A,TO/A,HARD/S
```

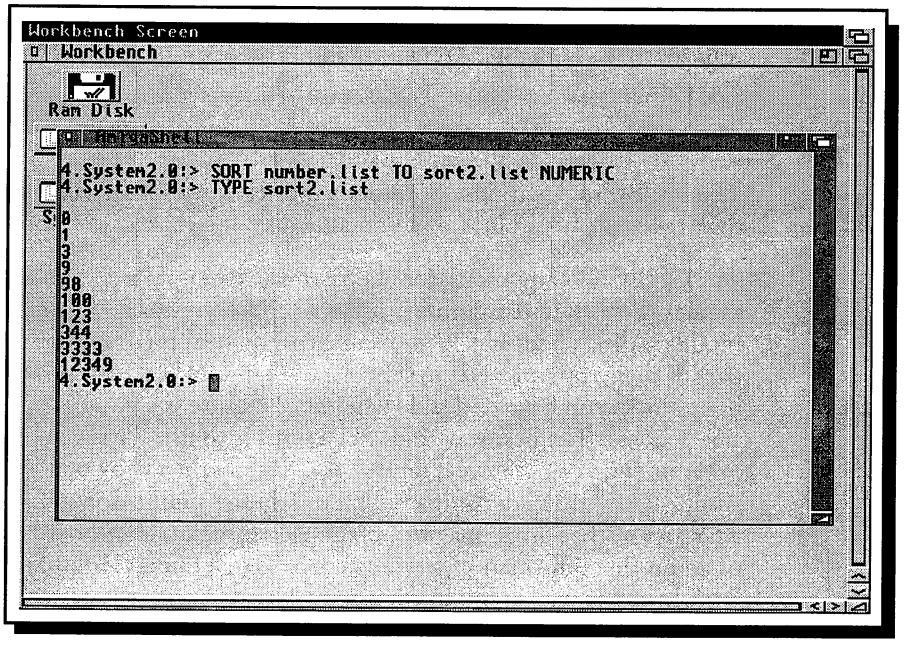


Figure 9-10 The NUMERIC Argument

By specifying NUMERIC as an argument, you force Sort to treat numeric characters as numbers, instead of as characters.

The FROM/A argument is the link file and cannot be a previously existing file. The TO/A argument is the linked file. Files that are linked must be on the same volume. Commodore's documentation states that the command creates "soft" links — the default — that can exist across volumes. This is incorrect; the command only creates links between files on the same volume, what the manual describes are "hard" links accessible using the HARD/S keyword. According to a source at Commodore, the MakeLink command does not work correctly in the first release of Workbench 2.0. It will probably be fixed in later releases, but, for now, I would avoid the MakeLink command.

Install: Making a Volume Bootable

When you initialize a floppy disk with the Format tool, you prepare the disk to store Amiga program and data files. The Install command goes one step further; it sets up a disk so that you can boot your Amiga with it. Install's template follows:

DRIVE/A,NOBOOT/S,CHECK/S,FFS/S

DRIVE/A is the device name of the floppy drive that contains the disk you want to make bootable. To make a floppy disk in the internal drive bootable, you enter:

```
INSTALL DF0:
```

The NOBOOT/S argument reverses the action of the Install command; it overwrites the boot block of a bootable disk and makes it nonbootable.

The CHECK/S argument does not write anything to the disk. Instead, it examines the disk to determine if it has a valid boot block. Some virus programs modify a disk's boot block, and some commercial programs do the same as part of their copy-protection schemes. When CHECK/S reveals a nonstandard boot block, you have to decide whether the cause is a virus or a copy-protection scheme. The best way to detect a virus is to use VirusX, a public-domain program written by Steve Tibbets. You can obtain a copy of VirusX from an Amiga users' group, a company that distributes public-domain programs (they advertise in the Amiga magazines), an electronic bulletin-board system (BBS), or an on-line information service such as PeopleLink, CompuServe, GEnie, or BIX. Install is very useful in combating viruses because it destroys boot-block viruses.

The FFS/S argument lets you install the boot block of the FastFileSystem instead of the default, which is the old file system (OFS).

DiskDoctor: Repairing Damaged Volumes

When you insert a disk into a drive, AmigaDOS attempts to validate the disk. If the disk validator finds errors, it reports that the disk is unreadable. (A common cause is ejecting a disk from the drive while the drive's disk-activity light is on.) To recover as many files from the damaged disk as possible, you use the **DiskDoctor** command, which has the template:

```
DRIVE/A
```

Unlike some commercial and public-domain disk recovery utilities, DiskDoctor writes to the damaged disk. Because of this, you may want to try another utility such as the public-domain DiskSalv before using DiskDoctor. DiskSalv does not change the damaged disk, but tries to copy as much information off it as possible. You can run DiskDoctor after you've run DiskSalv, but DiskSalv is useless if you've already run DiskDoctor on a disk.

When you run DiskDoctor, it prompts you to put the damaged disk in the drive. It then reads every cylinder on the disk, trying to recover all the files. When it recovers a file, it prints the filename. It also prints the location of any

errors it finds on the disk. If DiskDoctor is unable to recover the root block of the disk, it renames the disk as "Lazarus." When DiskDoctor is finished with a disk, you should immediately copy all the recovered files to a good disk. You can then try DiskDoctor on the disk again, but your chances of recovering more files are slim. After treating a disk with DiskDoctor, you should reformat the disk (with the normal option, not the quick option!) before using it again. If the disk doesn't work correctly after reformatting, throw it out.

The AmigaDOS Editors

Besides the MicroEMACS editor in the Workbench Tools drawer, two other text editors, Edit and Ed, come with your Amiga. You can access both through AmigaDOS. Both of these programs are heavily documented in the *Using the System Software* manual from Commodore, so I won't repeat that discussion here. Instead, I'll provide an overview of the philosophy behind each editor and detail some of the points the Commodore documentation overlooks.

The Edit Command

A line-oriented editor, Edit lets you edit a source file one line at a time, although it does support some global actions. The advantages of Edit are that it can work with a file of any size and that it can edit binary files. The disadvantages to Edit are legion. You can't scroll around in a file, positioning your cursor to make changes in the text. Edit doesn't have a cursor, and you can only have one line of the file available for editing at a time (see Figure 9-11). You can only *see* more than the current line by entering a special command!

To move from one line to another, you enter commands such as N (next), P (previous), and M*n* (move to line *n*). Once you move to the line you want to edit, you still can't directly change anything on the line. For example, say the current line contains a misspelled word, such as "computir." To change the "i" to an "e", you have to enter the command:

```
E/i/e/
```

To insert the word "Red" between the words "Go Sox" on a line, you have to enter:

```
A/Go /Red /
```

or

```
B/ Sox/ Red/
```

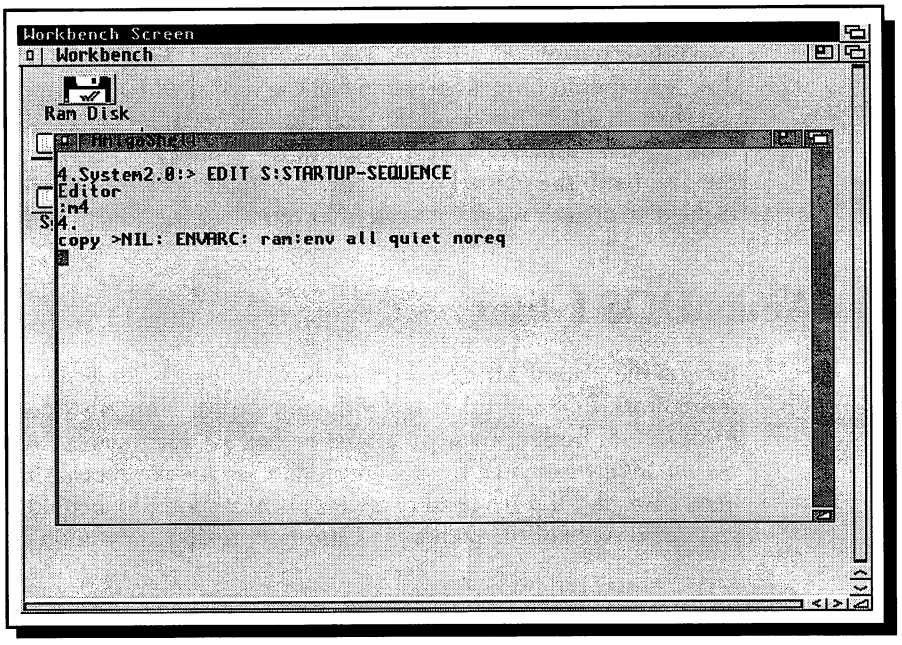


Figure 9-11 The Edit Editor

Because it is line-oriented, Edit only lets you work on one line of a file at a time. Even then, you can't edit the line directly; you must use awkward commands.

Too hard to learn and use, line-oriented editors went out of style in the 1970s and were replaced by screen-oriented editors. The Edit command is unsuited for creating the script and ARexx files you'll be using with your Amiga. If you need an editor that can handle files of any length and that can input binary data, check out a commercial text editor such as TxEd from MicroSmiths or CygnsEd Pro from ASDG. As far as Edit is concerned, I advise that you ignore its existence completely. It's the first command I delete from my system disk when I want to free up space.

The Ed Command

A step up from Edit, Ed is a screen-oriented editor. With Ed, you can scroll around a text file using the cursor keys and position the cursor where you want to insert or delete text. Ed is far more intuitive than Edit and far easier to use. Figure 9-12 shows what Ed looks like when editing the Startup-sequence file.

Up to now, the biggest knock against Ed was that, on a machine that offers a superb windowing interface, it supported neither the mouse nor menus. AmigaDOS 2.0 changes that.

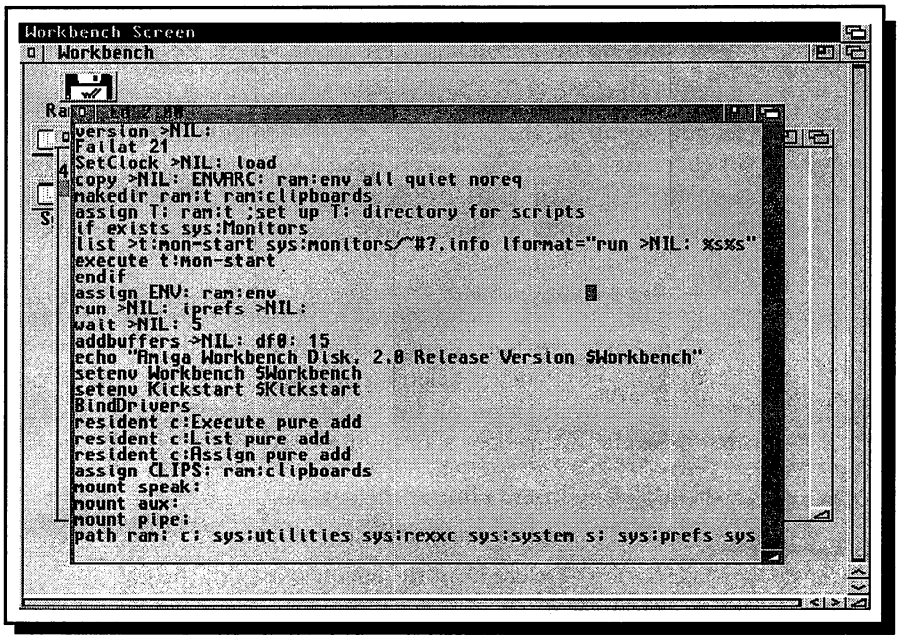


Figure 9-12 Editing with Ed

The Ed editor shows one screen of the file at a time, and lets you scroll around the file using the cursor keys. It also lets you edit text directly.

To start Ed, you enter its name on the command line, followed by the name of the file you wish to edit. If the file does not exist, Ed will create it. Thus, to create a file in S: called new-startup, you enter:

```
ED S:new-startup
```

Ed offers some other options on its command line as its template shows:

```
FROM/A,SIZE/N,WITH/K,WINDOW/K,TABS/N,
WIDTH=COLS/N,HEIGHT=ROWS/N
```

The SIZE/S argument lets you determine how much memory to set aside for the file you want to edit. The default is 60,000 bytes. Most command scripts and ARexx programs will be much shorter. The WITH/K argument designates a file containing lines of Ed commands. You can automatically edit the FROM/A file with Ed commands you have previously stored in the WITH/K file. If you don't designate a WITH/K file, Ed uses the ED-Startup file found in the S: directory.

The ED-Startup file is very interesting because it contains some undocumented commands, specifically SI and EM. The effect of these commands is to

reset the default Ed menus. (SI probably stands for Set Item, while EM might mean End Menus.) You'll learn more about the Ed menus below.

The WINDOW/K, WIDTH=COLS/N, and HEIGHT=ROWS/N arguments let you use an alternate console type (the default is CON:) and set the number of characters the window can display horizontally and vertically. The WINDOW/K argument takes the same form as it does with the NewShell command (see Chapter 8). The TABS/N argument sets the number of spaces in a tab stop; the default is 3.

Ed uses the cursor keys and mouse to position the cursor within a file, and has two types of commands — immediate and extended — to perform operations beyond simple character deletions and insertions. The immediate commands are available through simple keystrokes; to access extended commands, you first press the Escape key. Commonly used immediate commands are:

CTRL-A	Insert a line at the cursor
CTRL-B	Delete the line the cursor is on
CTRL-Y	Delete from the cursor to end of the line

When you press ESC, an asterisk appears at the bottom of the Ed window. This is your prompt to enter an extended command. Common extended commands are;

B	Move the cursor to the bottom of the file
BE	Cursor position is the end of a block of text
BS	Cursor position is the start of block of text
DB	Delete block of text marked by BE and BS
E/s/t/	Find the text s and replace it with t
F/s/	Find the text string s and position the cursor there
Q	Quit Ed without saving the file to disk
RP	Repeat the command; used with F and E
SA	Save the file to disk
T	Move the cursor to the top of the file
X	Save the file to disk and and quit Ed

A new aspect of Ed is that many of these commands are now available through menus. When you use the default ED-Startup, you gain access to the following menus: Project, Movement, and Edit.

Project: The items in the Project menu are Open, Save, Save As, About, and Quit.

Open: The Open item calls up the standard Amiga file requester to let you choose a file to edit. If you have already loaded a file, Ed lets you cancel the operation before you overwrite the file in memory.

Save: This item saves the file in memory to disk.

Save As: Save As calls up the standard file requester and lets you save the current file under a new name.

About: This item displays information such as the tab and margin settings, the start and end of the currently selected block, the size of the file buffer in bytes, and the percentage of the buffer that has been filled with text.

Quit: Quit lets you exit Ed without saving the contents of the buffer. When selected, it gives you a chance to change your mind before exiting.

Movement: The Movement menu deals with cursor positioning. Its items are Top, Bottom, Find, and Backwards Find.

Top: Moves the cursor to the top of the file.

Bottom: Moves the cursor to the bottom of the file.

Find: This item first prompts you to enter a string at the bottom of the Ed window. It then searches forward in the file from the current cursor position until it finds the specified string or reaches the end of the file.

Backwards Find: Almost identical to Find, this item searches for the string from the current cursor position to the top of the file.

Edit: The Edit menu consists of Delete Line, Query-Replace, and Redisplay.

Delete Line: This item deletes the line under the cursor.

Query-Replace: When you access Query-Replace, it prompts you for the string you want to replace in the file and then for the string you want to use in place of the first string. Ed then searches for the first string from the current cursor position. If it finds the string, it swaps in the second string.

Redisplay: Redraws the output window.

By putting many of the common Ed commands into menus, Commodore has made Ed much easier to use. The above menus and items are only a subset of the Ed menus. To see all the menus, you must first rename the ED-Startup file so that the Ed command can't find it:

```
RENAME S:ED-Startup S:Orig_ED-Startup
```

If you decide you don't want to use Ed with the extended menus, you can always rename the file back to ED-Startup or call it explicitly with:

```
ED my_file WITH S:Orig_ED-Startup
```

When Ed can't find the ED-Startup file, it opens with six default menus: Project, Edit, Movement, Search, Settings, and Commands.

Project: In addition to the items listed above, the default Project menu now includes New, Insert File, Write Block, and Save & Exit.

New: When you select New, Ed clears the file buffer (after asking you if your really want to do it). You can then begin entering a new file. The first time you save the file, you'll have to use Save As to give it a name.

Insert File: This item brings up the standard file requester, then inserts the contents of the file you choose into the file buffer at the current cursor position.

Write Block: Selecting Write Block brings up the standard file requester and lets you save the currently selected block as a file.

Save & Exit: This item saves the current file buffer to disk and exits Ed.

Edit: The default Edit menu bears little resemblance to the one defined by ED-Startup. The only item they have in common is Delete Line. The other Edit items are Undo Line, Start Block, End Block, Show Block, Insert Block, and Delete Block.

Undo Line: If you want to reverse changes made to the line where the cursor is positioned, you select Undo Line. Note that this item does not undo a Delete Line command.

Start Block: When you select this item, the current cursor position is marked as the start of a text block.

End Block: After marking the start of a block, you move the cursor to the end of the block you want to define and select End Block. The block definition stays in effect only as long as you make no changes to the text of the file.

Show Block: This item moves the start of the currently selected block to the top of the Ed window.

Insert Block: Copies the contents of the currently selected block to the cursor position. Using Insert block does not deselect the current block.

Delete Block: Deletes the text of the current block.

Movement: The default menu adds Go to Line, Next Page, and Previous Page to the Top and Bottom items defined in the ED-Startup Movement menu.

Go to Line: This item moves the cursor to the line number you designate. Ed doesn't display line numbers, so you'll have to use your best estimate.

Next Page: Selecting this item scrolls the text one page down in the file. A page is the number of lines currently visible in the Ed window.

Previous Page: Scrolls the text one page up in the file.

Search: The Search menu consists of Find, Find Next, Reverse Find, Reverse Find Next, Replace, Global Replace, Query Replace, and Global Q-Replace.

Find: Find prompts you for a search string and then moves the cursor to the next occurrence of that string in the file.

Find Next: Repeats the previous Find search, using the same string.

Reverse Find: Reverse Find lets you search for a string from the current cursor position towards the top of the file.

Reverse Find Next: Repeats the previous Reverse Find search, using the same search string.

Replace: Replace prompts you to enter a search string and a replacement string. Starting from the cursor position, it looks for the search string. When it succeeds, it replaces the search string in the file with the replacement string.

Global Replace: As with Replace, Global Replace prompts you to enter a search string and a replacement string. The only difference is that Global Replace replaces every occurrence of the search string with the replacement string from the cursor position to the end of the file.

Query Replace: This item is identical to Replace except that it asks you to confirm the substitution of the search string with the replacement string.

Global Q-Replace: Performs a global replace, but it asks you to confirm each replacement it makes.

Settings: The Setting menu has seven items: Set FN Key, Show FN Key, Reset Keys, Right Margin, Left Margin, Ignore Case, and Case Sensitive.

Set FN Key: This item lets you assign any Ed-extended command to a function key. You can see the current setting of a function key by selecting *Show FN Key*, and erase the settings of the functions keys with *Reset Keys*.

Right Margin: Right Margin lets you set the column number of the right margin. *Left Margin* does the same for the opposite side of the window.

Ignore Case: This item tells Ed to ignore capitalization when using the find and replace functions. *Case Sensitive* sets the opposite condition.

Commands: The Commands menu has five items: Extended Command, Repeat Last, Run File, AREXX Command, and Redisplay.

Extended Command: This item is the equivalent of pressing the Escape key; it brings up the Ed command prompt at the bottom of the window. You can then enter an extended command.

Repeat Last: This item repeats the last command you issued, whether from the command line or a menu.

Run File: When you access Run File, it brings up the Amiga file requester. You can then load and execute any file containing Ed commands. For example, if you renamed your ED-Startup file to Orig_ED-Startup to see all the Ed menus, you can reset the menus by calling up Orig_ED-Startup with Run File.

AREXX Command: This item lets you launch an ARexx script from inside ED. For more on ARexx, see Chapter 11.

Redisplay: This item forces a refresh of the Ed window.

If you are just starting out with Ed, you should use ED-Startup and the limited menus it provides. Often, the more options you have, the more confused things can become. As you become proficient with Ed, you can graduate to the default menus.

With the addition of menu and function-key support, Ed has become a far easier and more powerful text editor. Between it and MicroEMACS, you have two very good tools for creating and modifying AmigaDOS and ARexx scripts.

Working with Devices

When you talk about AmigaDOS devices, you are talking about a number of different things. An Amiga device can be a group of software functions that communicate with some peripheral hardware, such as the printer device — PRT: — which translates Amiga printer codes into the codes your printer understands. An Amiga device can also be a group of related functions that manage a software structure.The RAM: device redefines how the system treats a certain portion of memory, causing AmigaDOS to see some of your RAM as a disk drive.

AmigaDOS Devices are accessed and controlled by the device handlers in the L: directory, the device drivers in the Devs: directory, device drivers for expansion hardware that you copy to the Expansion drawer, and handlers and drivers built into Kickstart. Generally, you can divide AmigaDOS devices into those that are associated with the AmigaDOS filing system and those that are not. The former group includes all the floppy drives, hard-disk drives, and partitions on your system, and the RAM: device. The latter group includes

PRT:, SER:, PAR:, and SPEAK:, which deal with the peripheral ports, AUX:, RAW:, and CON:, which set up and communicate with console windows, and the PIPE: and NIL: devices, which handle program input and output.

Hard-Disk Devices: In the past, getting hard disks to work with AmigaDOS was a chore, because you had to define the characteristics of the drive in the system mountlist file and mount the devices in the Startup-sequence. These days, dealing with hard disks is much easier. Amiga hard-drive controllers, whether from Commodore or third-party manufacturers, adhere to a standard called Rigid Disk Blocks that stores the drive's characteristics on the drive itself and automounts the drive when you boot your system.

All hard-drive controllers come with preparation software like the HDToolBox that lets you format and partition the drive. Besides using supplied tools to prepare a drive, you may have to move the driver for your hard disk to the Workbench Expansion drawer. With autobooting and automounting hard-disk drives, however, you no longer have to worry about incomprehensible mountlist entries.

Floppy Disk Devices: Like the current generation of hard drives, floppy disk drives are automatically recognized and mounted by AmigaDOS when you boot your Amiga. The floppy drives are named DF*n:*, where *n* is a number from 0 to 3.

RAM: The RAM: device is a chunk of memory that acts like a disk drive. It provides fast storage but, because it is memory based, you lose its contents when you reboot your system. AmigaDOS automatically mounts the RAM: disk and gives it the volume name Ram Disk when you first access the disk.

PRT: The printer device lets you access and control your printer.

PAR: The PAR: device lets you output data through the parallel port.

SER: The SER: device lets you communicate with serial devices through the serial port.

SPEAK: With SPEAK: you can output text through the Amiga's voice output system as easily as you direct output to the printer. It must be mounted before use.

CON: The Shell uses the CON: device to control input and output for Amiga console windows. In addition to the features of the Shell described in Chapter 7, it also supports a wide variety of codes that let you control various aspects of the display. See the Echo command in the next chapter for details. CON: automatically mounts when you boot your system.

RAW: An alternative to CON:, the RAW: device is used by programs that want to work on the input from the keyboard directly. Like CON:, RAW: is automounted and always available.

AUX: The AUX: device provides unbuffered access to the serial port. By using it instead of CON: for the Shell console, you can use the serial port as a location for Shell input and output, allowing you to control a Shell from a terminal hooked up to the serial port. AUX: must be mounted before you can use it.

PIPE: The PIPE: device creates a direct link between the output of one program and the input of another. The PIPE: device must be explicitly mounted.

NIL: A catch-all device, NIL: is where you send output you don't care about. It is used a lot in command scripts to keep unimportant messages from cluttering the output window.

Most AmigaDOS commands access Amiga devices in one way or another; the following commands work directly with the devices: Addbuffers, Binddrivers, DiskChange, Mount, and RemRAD. Of these, Mount is by far the most important.

Mount: Adding Devices

The Mount command adds devices to AmigaDOS. It specifies the name of the device and any information AmigaDOS requires — such as the location of the device's driver. The template for the Mount command is:

```
DEVICE/A,FROM/K
```

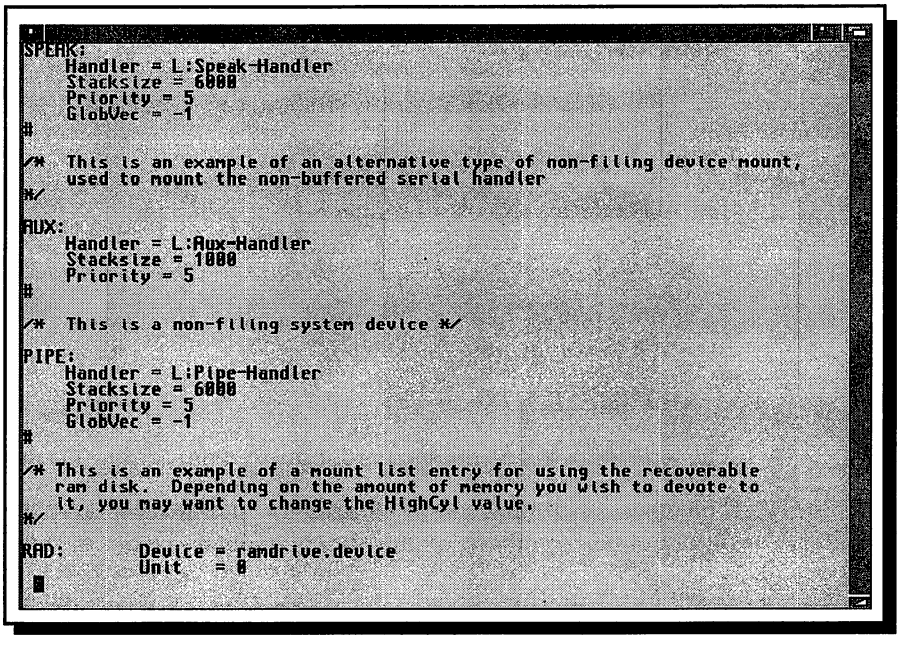
The DEVICE/A argument is the name of the device. This name must match the name of an entry in the Devs:mountlist file or in a mountlist file you designate with the FROM/K argument.

The standard mountlist for AmigaDOS 2.0 contains entries for four devices — SPEAK:, PIPE:, AUX:, and RAD: (see Figure 9-13). You can only mount the first three, however, because the driver for the RAD: device isn't present in the Devs: directory. For example, to mount the SPEAK: device, you enter:

```
MOUNT SPEAK:
```

To make the Amiga produce spoken output, you redirect output to the SPEAK: device. For example, to hear the mountlist file, you enter:

```
TYPE >SPEAK: DEVS:mountlist
```



```
SPEAK:
    Handler = L:Speak-Handler
    Stacksize = 6000
    Priority = 5
    GlobVec = -1
#

/*  This is an example of an alternative type of non-filing device mount,
    used to mount the non-buffered serial handler
*/

AUX:
    Handler = L:Aux-Handler
    Stacksize = 1000
    Priority = 5
#

/*  This is a non-filing system device */

PIPE:
    Handler = L:Pipe-Handler
    Stacksize = 6000
    Priority = 5
    GlobVec = -1
#

/* This is an example of a mount list entry for using the recoverable
   ram disk.  Depending on the amount of memory you wish to devote to
   it, you may want to change the HighCyl value.
*/

RAD:        Device = randrive.device
            Unit   = 0
```

Figure 9-13 The Mountlist

The Mountlist contains vital information about the devices you add to the system with the Mount command. Before the advent of Rigid Disk Blocks, knowledge of the Mountlist was vital to operate a hard disk. Now, you use utilities such as HDToolBox to prepare and partition hard-disk drives.

To control the output you hear, you can send options to the SPEAK: device by including the OPT keyword when you access SPEAK:. The options supported are:

p###:	Sets the pitch to a number from 65 to 320.
s###:	Sets the speed from 30 to 400.
m:	Sets the voice to male.
f:	Sets the voice to female.
r:	Gives the voice a robot-like quality.
o0:	Options can't be changed in the output stream.
o1:	Options are allowed in the output stream.
a0:	Use translator.library.
a1:	Don't use translator.library; output data are phonemes.
d0:	Use punctuation alone to break up sentences.
d1:	Breaks on-line feeds, carriage returns, and punctuation.

For example, to output the Startup-sequence file with a high-pitched, fast, robotic voice, enter:

```
TYPE >SPEAK:OPT/r/p300/s350 S:Startup-sequence
```

Options are separated by slashes.

The AUX: device also uses a simple Mount command:

```
MOUNT AUX:
```

To create a process that lets a terminal attached to the serial port run AmigaDOS commands, you enter:

```
NEWSHELL AUX:
```

The serial port will be the standard input and output for this process.

The other device you can mount from the mountlist is PIPE:, which uses the output of one command as the input of another. To do so, you simply enter:

```
MOUNT PIPE:
```

To use PIPE: you designate a file that acts as a conduit between two programs. For example, you can capture the output of a Dir ALL command in the Ed editor by entering:

```
RUN DIR >PIPE:my_file SYS: ALL
ED PIPE:my_file
```

The PIPE: file uses a 4,096-byte buffer between the program supplying output and the program using it for input. When the output program fills the buffer, it halts until the input program reads from the buffer.

The standard 2.0 Startup-sequence automatically mounts the SPEAK:, AUX:, and PIPE: devices. If you're tight on memory, you can edit the appropriate Mount commands from the Startup-sequence file to free up a little memory. Unless you obtain a commercial or public-domain device that you need to mount, you may never have to issue the Mount command from the keyboard.

AddBuffers: Adding Disk Buffers

Each disk drive attached to your Amiga has a buffer associated with it. This buffer speeds up access to and from disks by storing the most recently accessed disk blocks in RAM, where programs can read from and write to them faster.

The larger the size of the buffer, the faster the apparent access time to the disk. The Addbuffers command lets you add and subtract memory from the cache buffer of each drive. It has the following template:

```
DRIVE/A,BUFFERS/A
```

The DRIVE/A argument is the device name of the drive to whose cache you wish to add or subtract memory. BUFFERS/N lists the number of 512-byte chunks of RAM you want to add to or subtract from the cache. If the number is positive, the buffers are added; negative, and they are subtracted. You can't reduce the size of the cache to less than one buffer. For example, to add ten 512-byte buffers to the cache for your internal disk drive, you enter:

```
ADDBUFFERS DF0: 10
```

Adding buffers has a price; any memory devoted to disk buffers is not available to programs and data.

Binddrivers: Ties That Bind

In Chapter 5, I discussed the Expansion drawer, which holds device drivers for expansion hardware. The Binddrivers command matches the hardware with the driver that accesses it. Binddrivers has no template. It should be one of the first commands in the Startup-sequence file, where it can ensure that the system will be able to access and control your expansion hardware.

The DiskChange Command

Whenever you change a disk in one of the Amiga 3 1/2-inch floppy drives, the system automatically detects the change and updates its internal lists accordingly. The **DiskChange** command lets you inform the system when you've changed a disk in a drive that cannot automatically inform AmigaDOS about such changes.

What kind of drives are those? Any drive that lets you remove a disk and replace it with another may require that you use DiskChange. Examples are some of the new, high-density floppy drives, optical drives such as CD ROMs, WORMs, and magneto-optical read-write systems, and even the 5 1/4-inch drives that Commodore used to sell that let you read and write MS-DOS disks with the Amiga. These days, CrossDOS, a utility from Consultron that lets your Amiga floppy drives read and write 3 1/2-inch MS-DOS disks has just about eliminated the need for the Commodore 5 1/4-inch drive.

DiskChange has the following template:

```
DEVICE/A
```

To tell the system you've changed the disk in a drive, you simply enter the command, followed by the device name of the drive.

RAD: The Lost Command

Included in the Devs:mountlist is an entry for RAD:, a recoverable RAM disk. Unlike RAM:, the contents of RAD: can survive a warm boot. RAD: is unavailable under AmigaDOS 2.0, however, because Commodore did not supply its device handler on the release disk.

This of course makes the **RemRAD** command superfluous. Designed to remove the RAD: disk and return the memory it uses to the system memory pool, RemRAD now has nothing to remove. In the future, I hope Commodore either incorporates RAD: into Kickstart or supplies a new driver; RAD: is a useful device.

Conclusion

This chapter detailed the most important commands for manipulating the AmigaDOS file system and devices. In addition to accessing the commands in this and the previous chapter from the command line, you can also use these commands from within command scripts, the subject of Chapter 10.

– 10 –
AmigaDOS Command Scripts

An AmigaDOS script file resembles any ASCII text file on your Amiga system. You can create and edit it with Ed, see its contents with Type, and send it to your printer by copying it to PRT:. The major difference is that scripts consist of AmigaDOS commands. Script files let you automate tasks that would require multiple commands if entered into the Shell. The best example of a script file is S:Startup-sequence. Every time you boot your Amiga, AmigaDOS executes the contents of this file to configure your system. If you entered all the commands in Startup-sequence by hand, you'd have over 30 commands.

A script file is more than just a list of commands: it is a program. Commands are executed one at at time, starting from the top and moving sequentially through the script. Based upon the results of previous commands and user input, however, the script can decide to execute some commands and not execute others. Controlling the action of a script as it executes is the province of some special AmigaDOS commands.

The commands you'll learn about in this chapter are Ask, Echo, Else, EndIf, EndSkip, Eval, Execute, FailAt, Get, GetEnv, IconX, If, IPrefs, Lab, Quit, Set, SetEnv, Skip, UnSet, UnSetEnv, and Wait. Some of these can be used only in a script, others are available from the command line but most often used in scripts, and one, IPrefs, is here by default because it doesn't fit anywhere else. The most important script command is Execute.

Running Command Scripts

The most common way to execute a command script is with the Execute command. Execute takes a script file as input and oversees the execution of the

individual commands. It also lets you include arguments on the command line that are passed to the command script. Execute looks for the script file you designate in the current directory first and then in the S: directory, which is AmigaDOS' default storage location for script files.

Another way to execute a script is to call the script directly from the Shell. If a script file has its s bit set, you can execute it by simply entering its name on the command line. The Shell will search the current directory, the AmigaDOS search path, and finally the S: directory for the script file.

The NewShell and NewCLI commands also let you specify an AmigaDOS script that executes when you open a new Shell. (The ED-Startup file is *not* an AmigaDOS script, but a list of Ed commands.) Finally, from Workbench, you can execute a script by opening an IconX project (more on that later). Finally, you can execute a script by rebooting you system, which of course executes the Startup-sequence. No matter how you start them, however, AmigaDOS command scripts use the same commands.

The Execute Command

The Execute command doesn't have a template. It simply expects that the first argument passed to it will be the name of a script file. All other arguments are passed to the script itself.

You can create a script file in any text editor or from the keyboard. Enter the following:

```
COPY * TO S:my_script
DIR
CTRL-\
```

Now, to execute the script you've just created, enter:

```
EXECUTE my_script
```

The result of this one-command script is a listing of the current directory's contents.

Execute requires the use of one or more temporary files when it executes a script. It creates its temporary files in two places: the T: logical directory and the :t directory (:t means a physical directory named t located in the root directory of the current disk). The default Startup-sequence assigns T: to RAM:, so

you should have no trouble running Execute. If you ever get the following message:

```
Cannot create: ":t/Command-0-t01"
```

you'll know that Execute can't find a place to create its temporary files.

The Execute command lets you enter arguments for a script command on the command line. Just as command arguments let you control the actions of AmigaDOS commands, script arguments let you control the action of AmigaDOS scripts.

Why would you need to pass arguments to a script? Say, for example, that you created a script that uses the List command to capture the contents of a directory in a file and the Sort command to sort the filenames alphabetically. You called the script Alist (for alphabetical list) and stored it in S:. By entering

TYPE S:Alist

you now see the script's component commands:

```
LIST >list.file NOHEAD
STACK 20000
SORT list.file TO sort.list CASE
TYPE sort.list
DELETE sort.list list.file QUIET
STACK 4096
```

This script is simple and effective, giving you an alphabetical listing of files (see Figure 10-1), and cleaning up after itself. The problem is that it only works on the current directory. To list another directory, you have to move to that directory. By entering arguments on the command line, however, you can have the script work on any directory you designate.

Execute recognizes a number of parameter substitution statements in scripts. (A parameter is what you call an argument once it has been passed to its destination.) The most important is .KEY, which defines the parameter variables in the script.

A .KEY statement takes the form:

.KEY <parameter1>,<parameter2>,<parameter3>,etc.

The parameters are variable names used in the script. After parameter substitution, the variable name will stand in for the argument entered on the command line. To see how this works, look at what happens to Alist when you make some minor changes:

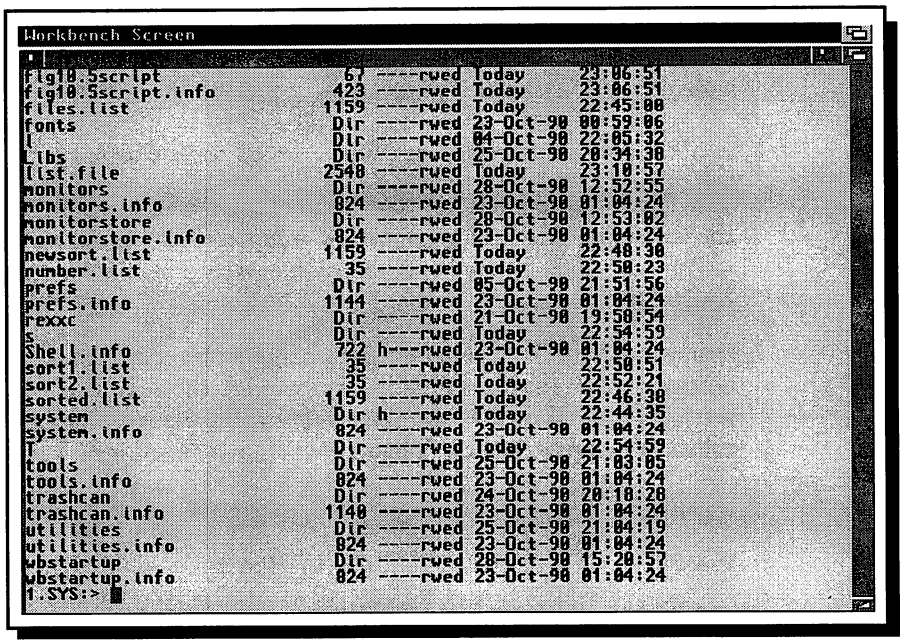


Figure 10-1 The Alist Script

One function of scripts is to create new "commands" by combining the capabilities of two or more existing AmigaDOS commands. Our Alist script, for example, combines the alphabetized listing of Dir with the information provided by List.

```
.KEY directory_name
LIST >list.file <directory_name> NOHEAD
STACK 20000
SORT list.file TO sort.list CASE
TYPE sort.list
DELETE sort.list list.file QUIET
STACK 4096
```

The variable "directory name" stores the first parameter passed to the script file by Execute. In this case, it will be the name of the directory you want to list. On the next line, the parameter name appears within brackets. When Execute sees this, it substitutes the value of the parameter for the parameter name. Thus, the argument you enter on the command line winds up in the parameter variable called directory_name. When this variable is accessed by a command, the command actually receives the value of the variable, which is the argument entered on the command line. For example, if you enter:

EXECUTE Alist SYS:Utilities

Execute passes the argument Sys:Utilities to the parameter variable directory_name. When directory_name is used by a command, the command actually receives the variables value, Sys:Utilities. Thus, when it finally executes, the List command looks like this:

```
LIST >list.file SYS:Utilities NOHEAD
```

and, in combination with the rest of the script, produces the output shown in Figure 10-2. This shows how you can use a script file to make a custom utility.

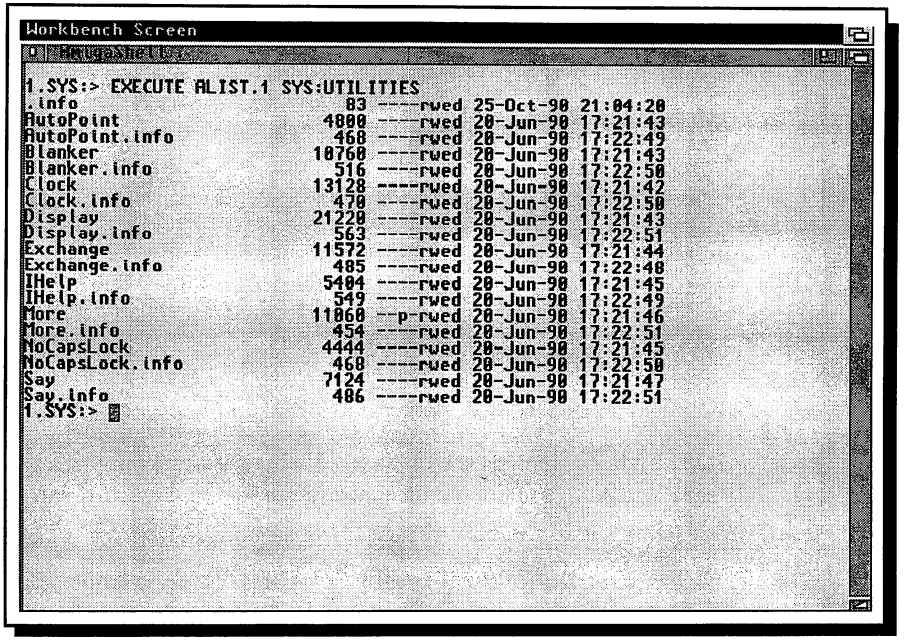


Figure 10-2 Arguments for Scripts

The .KEY keyword in a script lets you pass an argument from the command line into a variable within the script. In this case, the argument is the name of the directory (SYS:UTILITIES) you wish to list with the result shown here.

If you don't include an argument for the directory_name parameter on the command line, the parameter takes the value of a blank space. In the example, it would result in the listing of the current directory. You can define multiple parameters with the .KEY statement; they must be separated by commas only, no spaces. You can also have more than one .KEY statement in a script, but only the first one is recognized. Of course, the .KEY statement must appear before parameter variables in the script.

If you use multiple parameters in a script, Execute expects you to enter the arguments on the command line in the same order the parameters are defined in the key statement. You can alter the order of the parameters only if you precede the arguments with the names of the parameters. For example, if you have a script named My_script with the following .KEY statement:

```
.KEY param1,param2,param3
```

it expects the arguments on the command line to be in the same order: ALL, RAM:, and Sys:. You can change the order of the arguments by matching them with the parameter names on the command line. For example:

```
EXECUTE My_script param3 SYS: ALL RAM:
```

alters the sequence of arguments by explicitly matching param3 with the first argument. This is similar to the way you can use keywords to alter the order of arguments for an AmigaDOS command.

Besides .KEY, Execute recognizes several other keywords.

.DOT *s:* Changes the character that precedes the Execute keywords to *s.*

.BRA *s:* Changes the character that precedes a parameter variable in a script to *s.*

.KET *s:* Changes the character that follows a parameter variable to *s.*

.DOLLAR *s:* Changes the character that separates a parameter variable from its default value to *s.* The default delimiter is $. The keyword can be abbreviated to .DOL.

.DEF ***parameter "value":*** Assigns the string *"value"* to the parameter variable *parameter.*

The last two items point out how you can assign default values to parameter variables. Once a parameter is defined with the .KEY statement, you can define its value with .DEF. For example:

```
.KEY volume
```

defines a parameter called volume, and

```
.DEF volume "DF0:"
```

assigns the value DF0: to the variable. The default value is used if you don't include an argument for the parameter on the Execute command line. Another way to define a default value is when the variable is used in the script. If you've defined a parameter named device_name, you can assign the value PRT: to it in the command script by separating the variable name from the default with a

$, as in <device_name$PRT:>. As with the .DEF statement, the default in used when no argument is passed for that parameter.

Conditional Statements

AmigaDOS supplies a number of commands that let you test for conditions in a command script and to take different actions based upon the results of the test. You use the command **If** to create the test conditions, and the **EndIf** and **Else** statements to set the branches.

If sets up a simple test, the answer to which will be true or false. It takes two basic forms:

```
IF <condition>
<TRUE commands>
ENDIF
```

and

```
IF <condition>
<TRUE commands>
ELSE
<FALSE commands>
ENDIF
```

In the first form, when the condition tested is true, the TRUE commands are executed; when false, the script continues with the command after the EndIf command. In the second form, the TRUE commands are executed if the condition is true, and the FALSE commands are executed if the condition is not true. The conditions you can test for are found in the template of the If command:

NOT/S,WARN/S,ERROR/S,FAIL/S,,EQ/K,GT/K,GE/K,VAL/S,EXISTS/K

(See "Keyword Modifiers" at the end of Chapter 7 for definitions of /S, /K, etc.)

Perhaps the most common condition tested for in command scripts is whether or not a file exists. Such tests take the form:

```
IF EXISTS filename
```

One example of how to use this test is in the standard Startup-sequence script for Amiga OS 2.0. Recall that on hard-disk-equipped Amigas, the Tools directory is on the system disk, while on floppy-based machines, it is on the Extras2.0 disk. On a hard-disk system, the Startup-sequence wants to add Tools

to the search path. Here is how it does so without generating an error on floppy systems:

```
IF EXISTS SYS:Tools
PATH SYS:Tools ADD
ENDIF
```

On a hard-disk system, the condition is true and the Path command is executed. On floppy systems, the condition is false and the script avoids an error by jumping over the Path command.

The WARN/S, ERROR/S, and FAIL/S conditions refer to the AmigaDOS return codes. Whenever an AmigaDOS command finishes executing, it sets a return code that you can test. 0 means that the command executed normally. If the command encounters an unexpected condition but can still exit normally, it sets the return code to 5, equivalent to a warning. For example, if the Which command can't find the command you specify, it sets the return code to 5. The same happens if you ask the Assign command if a particular logical directory exists and the answer is no.

If a command encounters an error, it returns a 10, and if it fails completely, it returns a 20. For example, if you try to have Type output a file that doesn't exist, it returns a 10; if you neglect to pass it an argument, it returns a 20.

The return codes 5, 10, and 20 correspond to the WARN, ERROR, and FAIL keywords. You use the keywords to see if the previous command returned one of the error codes. If the return code is equal to or greater than the tested condition, the test is "true".

For example, if you have the Which command look for a particular command on the path, you can test the results and take appropriate action:

```
WHICH >NIL: Display
IF WARN
ECHO "Display program not found on search path"
ELSE
DISPLAY mypicture
ENDIF
```

With this script, if a return code of 5 (WARN) or higher occurs, the message appears on the screen. The FailAt command is often used in conjunction with the If command. FailAt sets the level at which an error will cause an entire script to abort. It remains in effect for one command only, so you should use it just before executing a command that may fail. For example, if you wrote a script that tries to type a nonexistent file, you will get a return code of 10. It won't do you any good to test for this code, however, because by default, script files abort if any command in them returns a code of 10 or better. To change

this default, you use the FailAt. For example, to continue running a script in the event that Type doesn't find a file, you include a FailAt command in your script just before the Type command:

```
FAILAT 11
TYPE myfile
IF ERROR
ECHO "File not found"
ENDIF
```

Because the fail level was set at 11, the script will not abort if Type can't find myfile. The next line checks the return code set by the Type command and prints an appropriate message.

The EQ/K, GT/K, and GE/K keywords let you test whether one string or variable is equal to, greater than, or greater than or equal to another string or variable. For example:

```
IF <parameter1> EQ "ALL"
```

To have the items compared as numeric values instead of strings, you include the VAL/S keyword. The NOT/S keyword negates the result of whatever comparison you're performing. For example,

```
IF NOT EXISTS myfile
```

is true if myfile does not exist. Similarly,

```
IF "ALL" NOT EQ <parameter1>
```

is true if parameter1 does not contain the word "All." You can use NOT/S to make up for some of the logical operations that aren't represented in the If keywords. For example, NOT GT is the equivalent of less than or equal to, and NOT GE is the same as less than.

Environment Variables

Besides using variables created with the .KEY statement, command scripts can also access environment variables, which come in two types: local and global. A local environment variable is only accessible from within a specified process; global environment variables are accessible from all processes running on the system.

What are environment variables? They are values that describe and define your current computing environment. Every AmigaDOS process has some default environment variables that describe the process. In the Startup-sequence,

AmigaDOS sets two global environment variables, Kickstart and Workbench, to the version numbers of the Kickstart and Workbench used by your computer. Finally, you can set your own local and global environment variables.

Six commands deal with environment variables: Get, Set, and Unset deal with local environment variables and GetEnv, SetEnv, and UnSetEnv deal with global variables.

Local Variables

Whenever AmigaDOS creates a process, it sets up three environment variables for the process: Process, RC, and Result2. Process contains the process number of the process, RC contains the result code returned by the last command that ran in the process, and Result2 contains the number of any exception or error created by the last command. To see the values of the local environment variables, you enter:

```
SET
```

AmigaDOS returns the short list shown in Figure 10-3. These are the values of the variables for the current process. If you create a new process with NewShell or NewCLI, it will have its own local variables.

The template of the Set command is:

```
NAME,STRING/F
```

As you've seen, when used without an argument, Set produces a list of the local variables. When used with arguments, it lets you set local variables. For example,

```
SET myname Bob Ryan
```

creates the environment variable name and sets its value to "Bob Ryan." When you enter Set without arguments again, the new variable and its value are listed.

You can also set two local variables by executing the Version command, which automatically creates two local variables — Kickstart and Workbench — and sets their values to the current version numbers.

You can recall environment variables in two ways, with the $ operator or with the Get command. Using the $ operator, you can use the variable from within a command. For example, if you want to print the value of the myname variable in the Shell window, you enter:

```
ECHO $myname
```

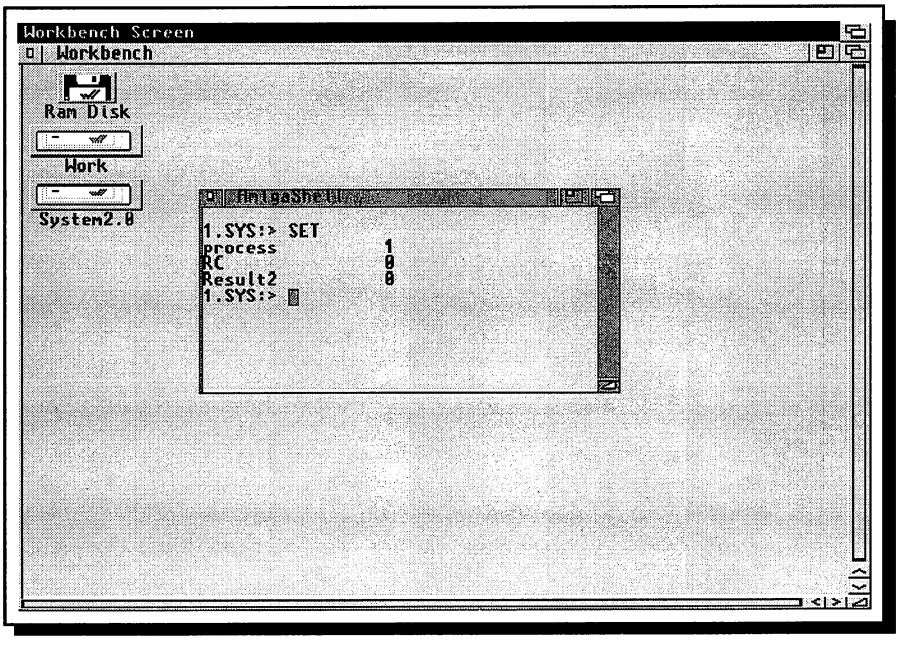


Figure 10-3 The Set Command

In addition to letting you create local variables, Set also displays any you've created plus the three — Process, RC, and Result2 — that are created when you open a Shell.

The system responds:

```
Bob Ryan
```

Whenever $myname appears on the command line or in a script file, the value of the variable myname is used in the command. (The Echo command is described later in this chapter.)

The Get command retrieves the value of the variable without using the $ operator. For example, if you enter:

```
GET myname
```

AmigaDOS responds

```
Bob Ryan
```

If you use the $ operator and Get together, you get some interesting results. For example, let's say you've created a variable my_var and assigned it the

value my_name, and you've also created a variable my_name and assigned it the value Bob. When you enter:

```
GET $my_var
```

you get the response:

```
Bob
```

Because both $ and Get return the contents of a variable, $ and Get together return the contents of the contents of a variable.

To erase a local variable, you use the UnSet command. When you enter it without an argument, it simply lists the local variables. When you enter it with the name of a variable, it deletes the variable and its value. For example:

```
UNSET myname
```

deletes the variable myname. You can delete any local variable you set up or any one set by the system. Whenever you close down a process, you erase the local variables.

Note that any local variables you create in one process are inherited by any Shells you spawn from the original process.

Global Variables

The global counterparts to Get, Set, and Unset are GetEnv, SetEnv, and UnSetEnv. These three commands deal with variables that are available to all processes running on the system.

SetEnv lets you set global environment variables available to all processes. It has the same template as Set. To create a global variable named model and give it a value "Amiga 3000," you enter:

```
SetEnv model Amiga 3000
```

When you enter SetEnv without an argument, you get a list of the *local* environment variables, not the list of global variables you'd expect. To see the global variables, you simply get a directory listing of the Env: directory, which is normally assigned to RAM:Env. All global variables are stored in Env:

The GetEnv command and the $ operator both retrieve global environment variables. The UnSet command deletes an environment variable.

Global variables are particularly useful in command scripts, where you can use them as loop counters. For an example, see the Eval command below.

Echo: Printing to the Screen

Oftentimes when you're executing a script, you want to print information to the screen. This could be information about an error or exception encountered in the script or the result of an action taken by the script. The Echo command provides this capability.

Echo has the template:

,NOLINE/S,FIRST/K/N,LEN/K/N

Echo outputs a string to the Shell window. For example, if you enter:

ECHO "This is a test of Echo."

you'll see:

```
This is a test of Echo.
```

in the Shell window. Echo prints what is between the quotation marks as a literal string. Echo can also output local and global environment variables. For example, if you are in process 1 and enter the following:

ECHO "The current process is number: " $process

your Shell will read

```
The current process is number 1
```

The NOLINE/S argument suppresses the carriage return at the end of the line. This is useful when you're prompting for input for the Ask command (more on this later). The FIRST/K/N and LEN/K/N arguments let you output part of a string only. The former defines the first character in the string that is output, and the latter determines the length of the output. For example:

ECHO "To be or not to be" FIRST 10 LEN 9

results in

```
not to be
```

Console Control Characters

One of the most interesting uses for Echo is sending console-control characters to the CON: device. Console-control characters let you make changes to the console device such as clearing the window, changing the type style to bold or italic, and using inverse characters for output.

Console-control characters come in two flavors: immediate commands and escape-sequence commands. In Chapter 7, you learned about the immediate commands involved in command-line editing. One more you should know about is CTRL-L, the form feed character. This command clears the Shell window and moves the cursor to the top of the window.

The most interesting console characters are the escape-sequence commands, which are so called because they are all preceded by the escape character. You can send these to the console device with the Echo and Prompt commands. Because the escape character is nonprintable, the console device recognizes the character combination *E as the escape character when it appears within a string. The console device also recognizes *N as the newline character.

There are about 20 console-control escape sequences, but most are rarely used formatting commands that are of little interest to most people. I've listed those commands, with short comments, in Figure 10-4. The two most interesting escape sequences are:

*Ec This sequence resets the graphics mode to normal and clears the screen.

*E[*n*;*n*;...*n*m This sequence sets the graphics rendition mode, where *n* is any number of the following:

0	set type style to plain text
1	set type style to boldface
3	set type style to italic
4	set type style to underline
7	use inverse type
3*n*	set foreground text color to *n*, where *n* is a number from 0 to 7
4*n*	set background text color to *n*, where *n* is a number from 0 to 7

To move down two lines, set the text style of the Shell window to bold italic inverse, print a message, reset to plain text, and move the cursor down two more lines, you enter:

```
ECHO "*N*N*E[1;3;7m Hello, World *E[0m*N*N"
```

```
*E[<i>n<d>@          Inserts <i>n<d> spaces into the current line. If
                    <n> is absent, the default is 1.
*E[<i>n<d>A          Moves the cursor up <i>n<d> number of lines. The
                    default is 1.
*E[<i>n<d>B          Move the cursor down <i>n<d> lines; default is 1.
*E[<i>n<d>C          Move cursor <i>n<d> spaces to the right; default
                    is 1.
*E[<i>n<d>D          Moves cursor <i>n<d> spaces to the left; default
                    is 1.
*E[<i>n<d>E          Moves cursor to column 1, <i>n<d> lines down;
                    default is 1.
*E[<i>n<d>F          Moves cursor to column 1, <i>n<d> lines up;
                    default is 1.
*E[<i>l<d>;<i>c<d>H Moves cursor to line <i>l<d>, column <i>c<d>.
*E[J                 Erases the screen from the current cursor
                    position.
*E[K                 Erases the current line from the cursor to the
                    end.
*E[<i>n<d>L          Inserts <i>n<d> lines above current cursor;
                    default is 1
*E[<i>n<d>M          Deletes <i>n<d> lines; default is 1.
*E[<i>n<d>P          Deletes <i>n<d> characters under the cursor;
                    default is 1.
*E[<i>n<d>S          Delete <i>n<d> lines from the top of the window;
                    default is 1.
*E[T<i>n<d>          Delete <i>n<d> lines from the bottom of the
                    window; default is 1.
```

Figure 10-4 Console Control Sequences

The control sequences recognized by the console device let you format output in a Shell window. The sequences listed here are often less used than the ones described in the main text.

Figure 10-5 shows the output of this command in a large font. The Shell-startup file (see Figure 10-6) is a script that is executed by default whenever you open a Shell. It contains three escape sequences to clear the Shell window, use reverse video, and return to normal output. It even gives you aliases to make access to these functions easier. To activate these options, use a text editor to strip the semicolons from in front of the Alias commands and save the file back to disk.

Ask: Getting Input

In addition to letting you output messages from a script, you can also input information to a script using the Ask command. The template of the Ask command is:

PROMPT/A

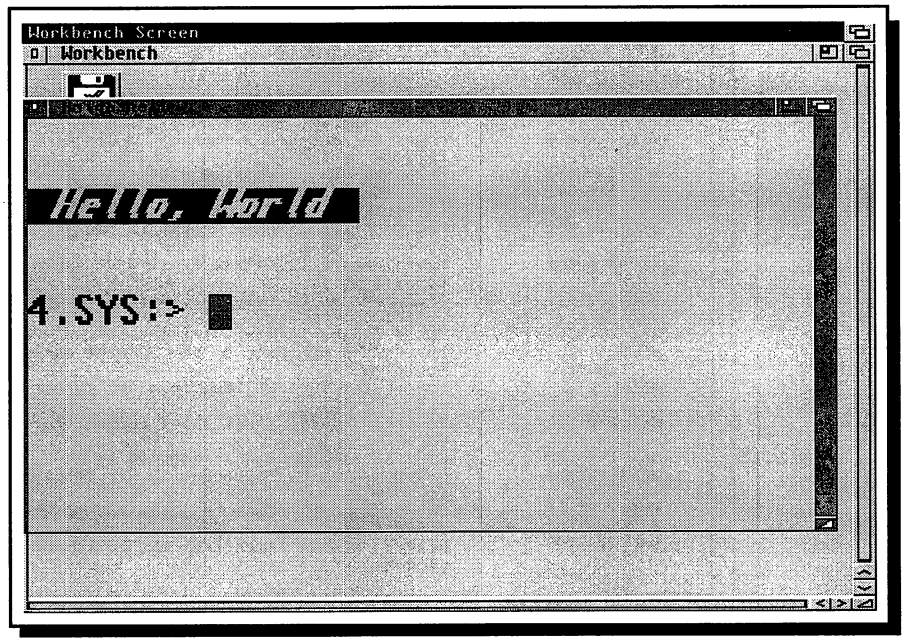


Figure 10-5 Escape Sequence Example

This sequence skips two lines, sets the type style to bold italic inverse, prints the message "Hello World," resets the type to plain text, and skips two more lines, all with one command (see text).

The PROMPT/A argument is a character string that prompts the user for input. The Ask command only accepts Y and N for input; it sets a return code of 5 if you enter Y, and 0 if you enter N or press the Return key. To test for input, you check the return code. For example,

```
ASK "Do You wish to Exit? "
IF WARN
QUIT 0
ENDIF
```

If you enter a Y in response to the prompt, the If condition is true and you execute the Quit command. Otherwise, you skip over the Quit command.

Another method of getting input for a script is described with the Skip command, in conjunction with labels.

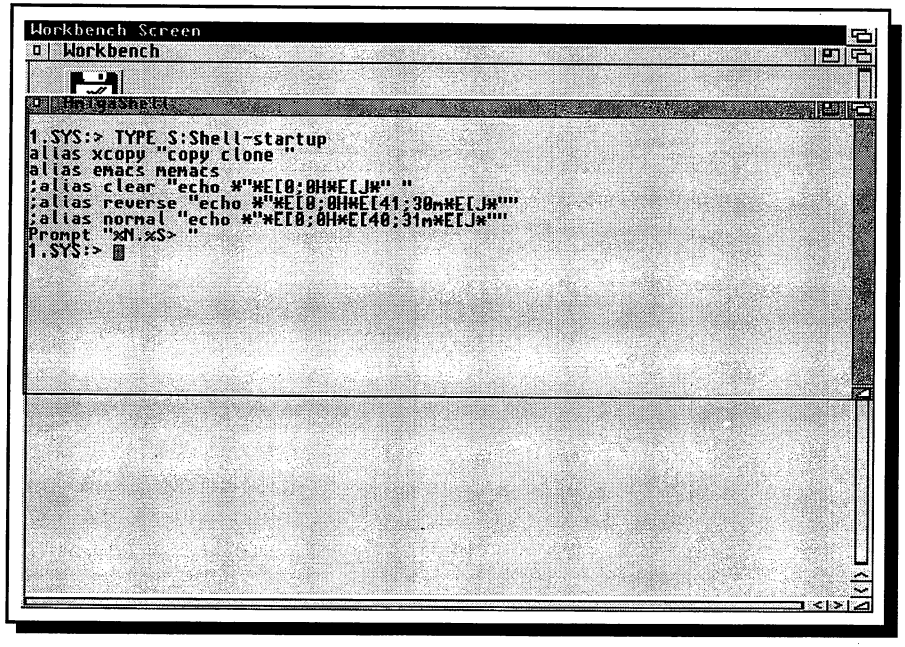


Figure 10-6 The Shell-Startup File

When you create a new Shell, this script file is automatically executed unless you designate another script with the NewShell FROM argument. By stripping the comment characters (;) from in front of the Alias commands, you make the clear, reverse, and normal functions available to you in your Shell sessions.

Using Labels in Scripts

You can label a specific point in a script using the Lab command, which helps you control the flow of the script. Lab acts as a target for the Skip command. It takes a single argument, a string of characters that are the label name. For example:

```
LAB Marker
```

establishes a label named Marker within a script.

You use labels in conjunction with Skip. When a script encounters a Skip command, it jumps forward to the label specified by the Skip. For example,

```
SKIP Marker
DELETE S:#?
```

```
DELETE C:#?
LAB Marker
```

causes the Execute command to skip from the Skip command to the label Marker; the two delete commands are not executed. If Skip doesn't find the specified label, it skips all the commands until it reaches the end of the script or encounters an Endskip command. Skip has the template:

```
LABEL,BACK/S
```

The BACK/S argument lets you tell Skip to search backwards through the script for its label. This is very important for creating loops that execute some commands again.

One of the most powerful features of Skip is its ability to accept your input while a script is running. If you use a question mark as the label in a Skip command, the command accepts input from the keyboard and skips to the label you enter. This is an excellent way to set up menus from script files. For example, if you have a script that creates a sorted list of files, you can ask where you want the list to wind up.

```
LAB Begin
ECHO "Enter P for Printer, S for Screen, A to append."
SKIP <* >NIL: ?
;
; Input is P
LAB P
TYPE sort.list TO PRT:
QUIT 0
;
; Input is S
LAB S
TYPE sort.list
QUIT 0
;
; Input is A
LAB A
IF EXISTS SYS:master_directory
JOIN SYS:master_directory sort.list TO SYS:mas_dir.temp
RENAME SYS:mas_dir.temp TO SYS:master_directory
ELSE
RENAME sort.list SYS:master_directory
ECHO "Master directory file created"
ENDIF
QUIT 0
```

```
;
; No valid input, loop to top
ENDSKIP
ECHO "Invalid input, try again"
SKIP BACK Begin
```

This script lets you input either a P, S, or an A, which instruct it to print your sorted list file, output it to the Shell window, or append it to a master directory file, respectively. Note that your input to the Skip command must be followed by a Return keypress. I've set off the different parts of the script with comments lines that begin with a semicolon (;). You include comment lines in a script to describe what is going on in case you want to change a script at a later date. Here, I use comments to make the script easier for me to follow.

Take careful notice of how the script handles bad input. If you enter anything other than P, S, or A, the Skip command falls through to the EndSkip command, from which, after printing a message, the script skips back to the Begin label. Note also the test made when you select the append option. If master_directory exists, the script appends your sort.list file to it; if it doesn't exist, the script makes sort.list the master directory file, ensuring that it will exist the next time you run the script.

Note also the use of the Quit command. This causes the script to exit and set the indicated return code. If you nest scripts, you can test this return value in a script that calls the exiting script. For example, to exit a script and return a warning, you use the command:

```
QUIT 5
```

The Wait Command

Often, you want to pause the execution of a script to give a command time to complete execution or to give the user time to read some output before continuing with a script. You may even want to wait for a specific time before executing a command. The Wait command handles these functions. It has the following template:

```
/N,SEC=SECS/S,MIN=MINS/S,UNTIL/K
```

By default, Wait suspends execution of the script for one second. If you enter a number, Wait suspends execution for that number of seconds. You can use the SEC=SECS/S keyword to make this clear in your scripts, although, because seconds is the default this keyword is redundant. You use the MIN=MINS/S keyword to set the wait time in minutes.

UNTIL/S waits until the system clock reaches the specified hour and minute. For example, if you have a telecommunications package named CompCom that you've set up to automatically access a stock quotation service and download information about your portfolio, you can initiate this procedure automatically at 5:30 PM with the following script:

```
WAIT UNTIL 17:30
CompCom
```

Eval: Evaluating Expressions

The Eval command lets you perform integer arithmetic in AmigaDOS. You can take actions within a script based upon the results of Eval. The command's template is:

```
VALUE1/A,OP,VALUE2,TO,LFORMAT/K
```

Eval requires only one argument. For example, if you enter:

```
EVAL 100
```

you get the response:

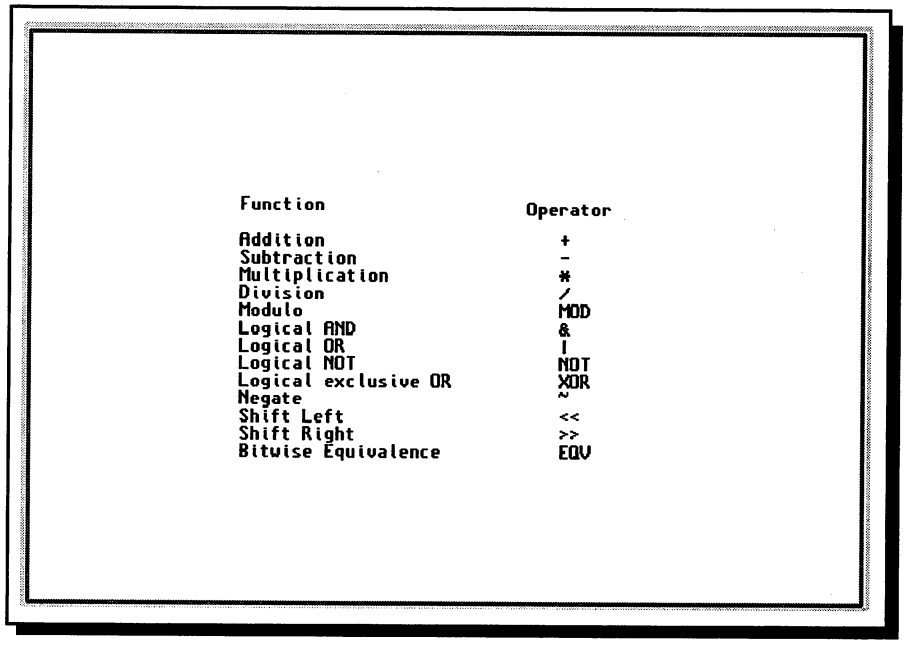
```
100
```

To get useful results out of Eval, you have to enter an arithmetic or logical expression using the OP argument. The keyword OP itself is optional and is rarely used. The operations performed by Eval are listed in Figure 10-7. Some example Eval expressions and their results are:

```
EVAL 12 + 6
18
EVAL 15 * 3
45
EVAL 13 / 3
4
EVAL 13 MOD 3
1
```

Note that Eval performs integer arithmetic only; if you enter a number with a decimal point, the part to the right of the decimal point gets truncated. If the answer contains a decimal, the decimal part is dropped, which is how 13 divided by 3 comes out to be 4. The MOD operator shown above gets its results

by dividing the first number by the second and then outputting the remainder of the division.



Function	Operator
Addition	+
Subtraction	-
Multiplication	*
Division	/
Modulo	MOD
Logical AND	&
Logical OR	|
Logical NOT	NOT
Logical exclusive OR	XOR
Negate	~
Shift Left	<<
Shift Right	>>
Bitwise Equivalence	EQV

Figure 10-7 Eval Operations

Although Eval supports a lot of functions, the simple arithmetic and logical functions are used most commonly. The bit-manipulation functions are rarely used.

Many of the Eval operations are designed primarily for programmers to use. The great majority of people have no need to OR or Left Shift a string of binary digits. Likewise, it will be rare that you use the LFORMAT/K argument, which lets you output the results of Eval in hexadecimal (%X), octal (%O), decimal (%N), or character (%C) format. (LFORMAT/K also supports the newline character (*N), which I always use to make sure the cursor moves down to the next line.) To convert the number 58, for example, from decimal to hexadecimal, you enter:

```
EVAL 58 LFORMAT "%X2 *N"
```

and Eval returns

```
3A
```

You specify the number of digits in the output by including a decimal number after the symbol for the type of output. You can also specify hexadecimal and

octal numbers for input. Hexadecimal numbers are preceded by 0x or #x; octal numbers are preceded by 0 or #. For example, to add a hexadecimal number to a octal number and see the results in decimal, you enter:

```
EVAL 0x2F + 017
```

and Eval returns

```
62
```

In addition to constants, you can also use parameter variables and environment variables in an Eval expression. For example:

```
.KEY input
.DEF input 10
SET constant 20
EVAL >ENV:answer <input> + $constant
ECHO $answer
```

This script outputs the number 30 if you don't include a value for input on the Execute command line. Note that by directing the results of Eval into Env:, you automatically create the global environment variable answer, which is available to other commands in the script. One use for this facility is to create looping structures. For example:

```
.KEY command
.DEF command DIR
SETENV loopcount 3
LAB Begin
EVAL $loopcount
IF EQ 0
QUIT 0
ENDIF
<command>
EVAL >ENV:temp $loopcount - 1
SETENV loopcount $temp
SKIP BACK Begin
```

This script runs the Dir command three times. (It will run some other command if you enter the command name on the Execute command line.) By using Skip BACK, it loops to the Begin label until the value of the loopcount variable is 0. Because loopcount then satisfies the test, the script exits. Note how every time through the loop between the Begin label and the Skip command, the value of loopcount decreases by one. Eval sets the value of temp to the current value of loopcount minus 1; the SetEnv command that follows resets loopcount to the value of temp. As you can see, Eval gives you a great

deal of flexibility in what commands your script executes and how many times it executes them.

The Startup-Sequence File

The most important script on your Amiga is the Startup-sequence file. It contains the commands that AmigaDOS executes whenever you boot your system. To illustrate how AmigaDOS scripts work and exactly what the most important script on you computer does, I'll go through the standard Amiga OS 2.0 Startup-sequence file, line by line. First, here is the file in its entirety.

```
VERSION >NIL:
FAILAT 21
SETCLOCK >NIL: LOAD
COPY>NIL: ENVARC: RAM:Env ALL QUIET NOREQ
MAKEDIR RAM:t RAM:clipboards
ASSIGN T: ram:t ;set up T: directory for scripts
IF EXISTS SYS:Monitors
LIST >T:mon-start SYS:monitors/~#?.info LFORMAT="RUN >NIL: %s%s"
EXECUTE T:mon-start
ENDIF
ASSIGN ENV: ram:env
RUN >NIL: IPREFS >NIL:
WAIT >NIL: 5
ADDBUFFERS >NIL: DF0: 15
ECHO "Amiga Workbench Disk. 2.0 Release Version $Workbench"
BINDDRIVERS
SETENV Workbench $Workbench
SETENV Kickstart $Kickstart
RESIDENT C:EXECUTE PURE ADD
RESIDENT C:LIST PURE ADD
RESIDENT C:ASSIGN PURE ADD
ASSIGN CLIPS: RAM:clipboards
MOUNT SPEAK:
MOUNT AUX:
MOUNT PIPE:
PATH RAM: C: SYS:Utilities SYS:Rexxc SYS:System S: SYS:Prefs
SYS:WBStartup ADD
IF EXISTS SYS:Tools
PATH SYS:Tools ADD
ENDIF
RexxMast >NIL:
```

```
IF EXISTS S:User-startup
EXECUTE S:User-startup
ENDIF
LOADWB
ENDCLI >NIL:
```

Now, let's look at the Startup-sequence file, line by line.

VERSION >NIL:

You might wonder why you'd want to execute the Version command, only to send the output to NIL:, the Amiga black hole device. The answer is that Version also sets two local environment variables — Kickstart and Workbench — when it executes. These variables carry the version numbers of your copies of Kickstart and Workbench.

FAILAT 21

This is for Amiga 500 owners who don't have battery-backed clocks. If you don't set the return code to above 20 and you don't have a battery-backed clock, the next command would abort the entire Startup-sequence.

SETCLOCK >NIL: LOAD

This command sets the system clock from the battery-backed clock. The only output SetClock sends to the screen is a carriage return. Because these aren't terribly aesthetic, they are sent to NIL:

COPY >NIL: ENVARC: RAM:Env ALL QUIET NOREQ

By copying the contents of Envarc: to RAM:Env, the Startup-sequence ensures that IPrefs and the Preferences editors can access the settings files. Also, this command creates the RAM: disk, which is mounted automatically the first time it is accessed. The ALL option ensures that the Envarc:Sys directory is also copied; QUIET, NOREQ, and the redirection to NIL: ensure that nothing is displayed to the screen while this command sequence is executing.

MAKEDIR RAM:t RAM:clipboards

The MakeDir command sets up some important directories on the RAM: disk.

ASSIGN T: RAM:t ;set up T: directory for scripts

Assigns the logical directory T to RAM:t. You need a T: directory before you can execute a command in a script that uses substitution.

IF EXISTS SYS:Monitors

Sys:, of course, refers to the boot disk. My Amiga 3000 came with a Monitors drawer; floppy-based systems do not. This command checks for the presence of the drawer and skips the next few commands if the drawer is absent.

LIST >T:mon-start SYS:monitors/~#?.info LFORMAT="RUN >NIL: %s%s"

Now we're having fun. This command creates a script file (note the redirection to T:mon-start). Each line of the script file reads "RUN >NIL: *pathname/filename*" The pathnames and filenames are those of all the monitor projects you've moved into the Monitors drawer.

Note the use of the tilde character (~) in the pattern matching wildcard. This character means NOT; it negates the rest of the search pattern. Thus, ~#?.info actually means match all the files that do *not* end with .info.

EXECUTE T:mon-start

Executes the script created with the previous command. Because it is the the default tool of all the monitor projects, the Run commands that make up the script actually execute AddMonitor.

ENDIF

Signifies the end of the If block. Systems without Monitor drawers pick up the script again with the following command.

ASSIGN ENV: ram:env

Determines the location of the global environment variables.

RUN >NIL: IPREFS >NIL:

Executes the IPrefs daemon, which remains active as long as the system is on. IPrefs monitors the Preferences settings in Env: and reports to Workbench when you change them with the Preferences editors.

WAIT >NIL: 5

This command gives IPrefs enough time to set itself up.

ADDBUFFERS >NIL: DF0: 15

Adds 15, 512-byte buffers to the default 5 allotted to DF0: to speed up disk access.

ECHO "Amiga Workbench Disk. 2.0 Release Version $Workbench"

Takes advantage of one of the local variables to report your Workbench version number. The copyright notice that appears before this message is built into Kickstart.

BINDDRIVERS

Looks in the Expansion drawer for device drivers that it can match to any autoconfig hardware attached to your system.

SETENV Workbench $Workbench

Sets a global Workbench variable from the local one.

SETENV Kickstart $Kickstart

Ditto for Kickstart.

RESIDENT C:EXECUTE PURE ADD

Adds an often-used command to the resident list. Note the use of PURE and ADD are optional; this usage is a holdover from AmigaDOS 1.3.

RESIDENT C:LIST PURE ADD

Adds another often-used command to those kept in memory.

RESIDENT C:ASSIGN PURE ADD

And another. If memory is very tight on your machine, you can remove the previous three commands from your Startup-sequence file.

ASSIGN CLIPS: RAM:clipboards

Sets the default location for clipboard information.

MOUNT SPEAK:

Forces AmigaDOS to recognize the speech device.

MOUNT AUX:

And the AUX: device.

MOUNT PIPE:

And the PIPE: device. If you want to save a little memory, and you rarely use these devices, delete the last previous commands.

PATH RAM: C: SYS:Utilities SYS:Rexxc SYS:System S: SYS:Prefs SYS:WBStartup ADD

Add a whole lot of directories to the AmigaDOS search path. The important thing to note here is the addition of the C: directory to the list. This may seem redundant, considering that C: is always in the search path. However, it is always the *last* item in the default path. Putting it closer toward the front of the list means you won't be searching a lot of directories before you search C: which, after all, is where most of your commands are. Also note that the ADD keyword is not required.

IF EXISTS SYS:Tools

This test keeps the Startup-sequence from aborting on floppy-based systems, where Tools is not on the boot disk but on Extras2.0.

PATH SYS:Tools ADD

Adds the Tools directory to the AmigaDOS search path for systems where Tools is on the boot disk. The ADD keyword is not required.

ENDIF

End of the If block. Floppy-based systems take up the script again with the next command.

RexxMast >NIL:

Starts up the ARexx resident process. Ensures that applications that use ARexx won't have to force you to load the process.

IF EXISTS S:User-startup

If you want to add commands to the Startup-sequence without altering the Startup-sequence file, you can create a file named S:User-startup and put the additional commands you want in there. If you don't create a User-startup file, you skip the next command.

EXECUTE S:User-startup

If the program passed the previous test, you must have a User-startup file. This command executes it.

ENDIF

Systems without a User-startup file rejoin the action with the next command.

LOADWB

Loads and runs the Workbench interface.

ENDCLI >NIL:

Ends the original Shell process. If you want to have a Shell active when you boot, you can delete this command.

That's it for our look at the Startup-sequence file. It contains many illustrative examples of how to write scripts — especially of how to deal with potential errors before they occur. Use it as a model of your own scripts, and don't be afraid to modify it. Just remember to rename the original Startup-sequence file to something like Orig-startup before you go with a modified startup.

IconX: Running Scripts From Workbench

Command scripts are not limited to AmigaDOS. The IconX (Icon eXecute) command lets you execute a script by double-clicking on an Icon from Workbench.

You access IconX from an IconX project. You create such a project by attaching a .info file to an AmigaDOS script file. See the section "IconEdit: A Practical Example" in Chapter 6 to learn how to attach project icons to icon-less files. Once you've created a project icon for your script file, you have to set the Default Tool in the project's Information window to C:IconX.

IconX projects can use two Tool Types, DELAY and WINDOW.

DELAY=: The DELAY Tool Type lets you enter a number that indicates how much time will elapse between the termination of the script and the closing of its output window. You can use it to keep the window open long enough to see the output of the script. If you enter:

DELAY=0

IconX will keep the window open until you enter CTRL-C. The DELAY value isn't in seconds. On my Amiga 3000, a delay of 100 lasted about 3 seconds.

WINDOW=: This Tool Type lets you specify the console device, position, and size of the output window created by IconX. The window information takes the form:

console.device/x/y/w/h/title

console.device: Lets you specify the console device to be used for the output window. You should stick to CON:.

x: This specifies the distance in pixels from the left-edge of the screen to the left-edge of the output window.

y: The distance in pixels from the top of the screen to the top of the output window.

w: The width of the window in pixels.

h: The height of the window in pixels.

title: The name of the output window.

Even if you don't plan to use IconX for your own scripts, you should not delete it from the C: directory. Many commercial programs use it for installation scripts.

The IPrefs Command

I stuck IPrefs in the last section of the last AmigaDOS chapter for a reason; it is unlike any other AmigaDOS command. Because of how it operates, it is in a class of programs called a daemon. When run, IPrefs remains active and in memory for as long as your machine is on, although you can close it down by sending it an attention flag using the Break command. It uses hot links to determine when the settings of the Preferences editors have been changed, and gets Workbench to update its display based upon those changes. IPrefs should always be called from your Startup-sequence file. I guess that's why I put it in this chapter.

Conclusion

We've reached the end of our discussion of AmigaDOS command scripts and AmigaDOS itself. As you become familiar with how commands and scripts work, you'll build scripts that combine commands into new commands. As long as you keep your important data files backed up, you shouldn't hesitate to experiment with commands and scripts. They greatly extend your power over your system.

– 11 –

Introduction to ARexx

Unlike Workbench and the Shell, which have been included in one form or another in every release of the Amiga OS, ARexx is new to the Amiga system software. ARexx is not, however, new to the Amiga; it has been available since 1987 from its creator William Hawes. The fact that Commodore is making it a permanent part of the operating system is a testament to Hawe's excellent work in bringing ARexx to the Amiga.

What is ARexx? The simplest answer is that it is a programming language. It lets you string together a number of instructions for your Amiga to execute. In this regard, it doesn't differ much from Amiga Basic, the language it replaced.

ARexx, however, is a different programming language: ARexx programs can communicate easily with other programs on your system and can send commands to these programs. For example, you can have an ARexx program issue AmigaDOS commands to the Shell. In effect, an ARexx program can integrate the capabilities of many different applications. ARexx lets you create meta-applications by combining the capabilities of many individual applications.

This chapter and the next introduce ARexx and provide examples of using the language. They are meant to supplement the reference information that Commodore provides in *Using the System Software.* They hit on the high points so that you can get started with ARexx quickly and easily, and give you enough background so that you can understand the Commodore documentation.

Note that I use lowercase characters when talking about ARexx instructions, to differentiate them from AmigaDOS commands.

ARexx Roots

ARexx is the Amiga version of Rexx, a program developed by Michael Cowlishaw of IBM for IBM mainframe computers. Cowlishaw had a number of goals in mind when he developed Rexx. Most importantly, he wanted a language that was easy to learn and use. Consequently, he limited the number of built-in instructions and eliminated the need to deal with data types. Every value in a Rexx program is a string and can be stored and manipulated as a string.

Cowlishaw developed Rexx between 1979 and 1982, and it became a part of the IBM VMS operating system in 1983. It has since found its way to many IBM platforms, including OS/2. It has also been ported to many other operating systems, such as MS-DOS.

Since William Hawes brought it to the Amiga, ARexx has enjoyed a steadily growing popularity, primarily because of its ability to tie multiple applications together. To be available to ARexx programs, however, an application program must contain an ARexx message port. At first, no Amiga programs had such a port. Over the past couple years, however, programs as diverse as Digi-Paint 3 from NewTek, ProWrite 3.0 from New Horizons, and MicroFiche Filer Plus from Software Visions have added ARexx support. With ARexx now an integral part of Amiga OS, you will see a lot more products supporting ARexx in the future.

Language Basics

To run ARexx programs on your Amiga, RexxMast must be active. The standard Startup-sequence for Amiga OS 2.0 loads RexxMast automatically, so it will be active on your system after you boot your computer. If you've altered the Startup-sequence, you can activate ARexx by opening the RexxMast icon or by entering from the Shell:

```
SYS:System/REXXMAST
```

If you're going to use ARexx, you should also make a logical assignment so that RexxMast knows where you've stored your ARexx programs.

```
ASSIGN REXX: SYS:Rexxc
```

tells RexxMast to look for your ARexx programs in the Sys:Rexxc directory.

To create ARexx programs, you need to use a text editor. For most ARexx programming, I use the Ed editor. For long, complicated programs, I use one of the powerful commercial text editors available for the Amiga. For most jobs, however, Ed is just fine.

Let's create a simple ARexx program. First, you have to call up Ed.

```
ED REXX:myprog.rexx
```

This creates the file myprog.rexx in the REXX: directory. You don't have to include the .rexx extension to the filename; but I and many other people use it as a reminder that the file is an ARexx program.

Once in Ed, enter the following:

```
/* An ARexx Program */
say 'The say instruction prints to the screen.'
say 'It prints anything between the quote marks.'
```

Now select the Save item from the Project menu, followed by Quit. (If you're using the extended Ed menus, you can select Save & Exit.) From the command line, enter the following:

```
rx myprog
```

The *rx* command passes the file myprog to ARexx, which interprets the contents of the file and produces the result:

```
The say command prints to the screen.
It prints anything between the quote marks.
```

You use rx to execute ARexx programs from the Shell command line. Notice that the ARexx program uses the same output window as the Shell process that called it.

Although the above program is simple, it illustrates two important points, besides describing in a nutshell what the say command does. First, every ARexx program must begin with a comment, which is defined as any text string between the comment delimiters /* and */. More importantly, it illustrates the fact that in ARexx, like all other programming languages, instructions are executed in the order they appear in the program. Only by using special instructions can you change the order of instruction execution.

Constants and Variables

All computer programs manipulate information, and ARexx is no different. It lets you manipulate data in the form of constants and variables.

A constant, also known as a literal, is simply a number or a string of characters. For example:

```
'Amiga'
931
'The HMS Titanic'
```

are all examples of literals. They are the data.

You can also store data in variables. As the name implies, the data stored in a variable can change. A variable has two components, a name and a value. Examples of variable names are:

```
Amiga
TheHMSTitanic
```

Variable names, also called symbols, can't begin with a numeral. They can contain any letter or numeral and periods (.), exclamation points (!), question marks (?), and underscores (_). To assign a value to a variable, you use the assignment operator (=). For example,

```
/* Using Variables */
Name = 'John'
Name_2 = 'Mary'
John = Name_2
say 'The name contained in John is' John
say
John = 5
Name = 10
say 'The sum of John and Name is' John + Name
```

When you run this program, you get the following results:

```
The name contained in John is Mary

The sum of John and Name is 15
```

This program illustrates some important principles. First, you use the assignment operator (=) to give a variable a value. On the left side of the operator is the variable name, on the right is its value. As the third line shows, the value you assign can be the contents of another variable; it can also be the results of

an arithmetic or string operation. In the fourth line, the say command printed the literal as it appeared in the program, and then printed the *value* of the variable John. Whenever a variable name appears in an instruction, the instruction always works with the value contained in the variable. After the second say instruction printed a blank line, the program assigned a number to the variable John, which had previously held a character string. Although this is strictly forbidden in most other languages, it is permitted is ARexx. You don't have to declare that a variable can only hold one type of data. ARexx variables can hold any type of data at any time. (The data itself is always stored as a string of characters. ARexx decides whether to treat it as numeric information or character information depending upon how the data is used in an instruction.)

In addition to simple variables that contain one value, ARexx lets you create compound variables that contain many values. A compound variable or symbol consists of a stem and a tail. For example, the compound variable

```
Name.number
```

consists of the stem Name. (including the period) and the tail number, while the compound variable

```
Name.george
```

has a tail named george. To assign a value to a compound variable, you use the assignment operator:

```
Name.number = "George"
Name.george = "Harry"
```

When you assign a value to the stem of a compound variable, you set the value of every instance of that stem to the value. For example:

```
Name. = 'Mary'
```

assigns the value 'Mary' to both Name.number and Name.george. Compound variables let you set up arrays of numbers and strings. More on these later.

Basic Operations

To store data in a variable, you use the assignment operator. To manipulate data in other ways, ARexx provides nearly two dozen arithmetic, string, and logical operators.

The most familiar operators are addition (+), subtraction (-), multiplication (*), and division (/). You use these to manipulate numeric data. For example:

```
/* Basic Math */
Num1 = 4
Num2 = 3
say 'the sum of the numbers is ' Num1 + Num2
say 'the product of the numbers is ' Num1 * Num2
say 'the quotient of Num1 divided by Num2 is ' Num1/Num2
say 'the difference between Num1 and Num2 is ' Num1 - Num2
```

This program prints:

```
the sum of the numbers is 7
the product of the numbers is 12
the quotient of Num1 divided by Num2 is 1.33333333
the difference between Num1 and Num2 is 1
```

You don't use arithmetic operators with the say instruction only; you also use it to set the value of variables. For example:

```
/* Miles Per Gallon */
distance = 493.5                    /* distance traveled in miles */
gas_consumed = 14.6                 /* gas used in U.S. gallons */
price = 1.65                        /* price per gallon */
mpg = distance / gas_consumed       /* miles per gallon */
say 'Our mileage for the trip was 'mpg 'miles per gallon'
ppm = (gas_consumed * price) / distance /* price per mile */
say 'and the price per mile was $'
```

The mileage, as computed by this program is just over 33.8 MPG and the price per mile is just under 4.9 cents per gallon. Note the use of comments to document the program, and the use of parentheses in the computation of the price per gallon. The parentheses ensured that the multiplication was carried out before the division.

The other arithmetic operators are exponentiations (**), integer division (%), and modulus division (//). For example:

```
/* Still More Math */
num1 = 10
num2 = 3
say num1 ** num2 num1 // num2 num1 % num2
```

returns the values

```
1000 1 3
```

The exponentiation operator returned 10 raised to the 3rd power, or 1000. The modulus operator returned the remainder of 10 divided by 3, while the integer division operator returned the integer portion of the same operation.

In addition to the operations that act on numeric data, ARexx has three operators for string data, all of which concatenate one string with another. The first operator is simply a blank space. If you use it between two variables containing character information or between a variable and a literal, you get a combination of the two with a blank in between. For example:

```
/* Combining Strings */
First_Name = 'George'
Name = First_Name 'Bush'
say Name
```

prints out

```
George Bush
```

The second operator (||) concatenates without an intervening space. The third concatenation operator is implied; if you don't leave a space between strings, whether literal strings or variables, ARexx concatenates them. For example:

```
/* Read My Strings */
first_name = 'George'
last_name = 'Bush'
name = first_name || last_name
name1 = first_name'Bush'
say name name1
```

gives you

```
GeorgeBush GeorgeBush
```

Note that the Commodore manual sometimes shows the concatenation character to be two Greek letters not available on an Amiga keyboard. Just remember that the operator is two vertical bars produced by the shifted-Backslash key.

Basic Input and Output

You've already seen how the Say command outputs information to the screen. You can input data to an ARexx program with the *pull* command. By not having to specify the value of every variable when you create a program, the pull

command lets you create far more general and useful programs. Compare the following program with the Miles Per Gallon program used above.

```
/* Son of Miles Per Gallon */
say 'The Automatic Mileage Calculator'
say 'Enter distance traveled'
pull distance                          /* enter the distance */
say 'Enter gasoline consumed'
pull gallons                           /* enter fuel consumed */
say 'Enter price per gallon'
pull price                             /* enter price of gas */
say
mpg = distance / gallons
ppm = (gallons * price) / distance
say 'Your car got ' mpg 'miles per gallon on this trip.'
say 'You paid ' ppm 'cents per mile.'
```

The most important difference between the two programs is their versatility. The first program calculated the mpg and ppm for one trip only. To change the variables, you have to change the program. The second program pauses to let you enter values for its variables from the keyboard as the program runs. You can use it to calculate your mileage and costs for any trip, making it a much more general and useful program.

The pull instruction is actually a special case of the more general parse instruction, which lets you manipulate strings in many different ways. Pull takes its input from the keyboard and, if the input is text, converts any lowercase characters to uppercase. In default mode, pull simply waits for you to enter data at the cursor. You can set a prompt for pull by using the options instruction. For example,

```
options prompt 'Enter input:'
```

will display

```
Enter input:
```

Whenever your program uses the pull command.

Branching Instructions

If all a programming language could do was execute instructions serially from the top of the program to the bottom, they wouldn't be of much use. A pro-

gram has to be able to make decisions based upon input it receives and results it calculates. With ARexx, the basic decision-making instruction is *if*.

Like the If command in AmigaDOS, the ARexx if tests a condition to see if it is true or false. The condition can be either a comparison of two values or a logical expression. The if instruction works with the then instruction and the else instruction to create logical control blocks in ARexx. To see how if works, run the following program.

```
/* If I Were a Rich Man */
say 'Input total of all assets'
options prompt 'Enter here:'
pull assets
say 'Input total of all debts'
pull debts
net = assets - debts
if net > 0 then say 'Wow! your net worth is $'net
if net = 0 then say 'Cutting it close, aren''t you?'
if net < 0 then say 'Money isn''t that important anyway'
```

Taking your input, the program tests the condition and prints the appropriate message. It has made a decision and executed the instruction that handled that condition. (Another item of note in this program is the use of the two single quotes in the second and third if clauses. These let you use the single-quote character within a string, instead of always having it delimit the end of the string. One single-quote character still delimits a string, but two together puts one single quote into the string.)

The above program is rather inefficient because it will test the equals and minus conditions even if the plus condition is true. You can get around this in a couple of different ways. For example:

```
/* More on if */
say 'Input total of all assets'
pull assets
say 'Input total of all debts'
pull debts
net = assets - debts
if net > 0 then say 'Wow! Your net worth is $'net
 else if net =0 then say 'You''re cutting it close.'
    else say 'Money can''t buy you love'
```

With this program, if the first condition is true, the other conditions are not tested.

You are not limited to one instruction after a then or an else instruction. You can execute a block of instructions by using the *do* instruction. For example:

```
/* If-Then-Else Blocks */
say '  Enter first number'
pull num1
say '  Enter second number'
pull num2
begin:
say 'Do you wish to:'
say '  Add two numbers (a)'
say '  Multiply two numbers (m)'
pull choice
if choice = 'A' then do
 sum = num1 + num2
 say 'The sum of ' num1 ' and ' num2 ' is ' sum
 end        /* end a true block */
else if choice = 'M' then do
 product = num1 * num2
 say 'The product of ' num1 ' and ' num2 ' is ' product
 end        /* end of m true block */
else do
 say 'You must enter an m or an a!'
 signal begin
 end
```

Remember that when you input a character or character strings with pull, the input is converted automatically to uppercase. So, even though you can enter a lowercase m or a, you must test the choice in the program with uppercase characters.

In addition to comparing two values with =, > (greater than), >= (greater than or equal to), < (less than), <= (less than or equal to), and ~= (not equal to), you can also test Boolean expressions. For example, the test

```
if flag
```

will be true if the value of flag is 1 and false if it is 0, while

```
if ~flag
```

sets the opposite condition. The following expression,

```
if (flag | count)
```

is true if either flag or count (or both) is 1, false if both are 0. The other Boolean operators available are & (AND) and && (Exclusive OR). Note that you can also use Boolean operators to combine comparison operators, as in:

```
if (count = 5) & (response = 'Y')
```

In the above program, the do and end instructions delimit the blocks of code that are executed based on the results of the condition tests. The signal instruction, which is only executed if neither an a or an m is entered, jumps the program back to the begin: label. A label is simply a string of characters that ends with a colon (:). It acts as a target for signal instructions.

Looping Structures

You usually use if-then-else blocks when you have multiple conditions to test and you anticipate executing the code just once. Combined with some other special instructions, the do instructions let you loop through a block of statements as many times as you like.

In loops, the do instruction takes three forms. In the first, do sets up a loop that you execute a predetermined number of times. For example, try this program.

```
/* Looping a Set Number of Times */
say 'This program loops five times'
do 5
 say 'Hello World'     /* every programming tutorial needs
                          one Hello World program */
 end
/* Looping a User-Set Number of Times */
say 'Enter the number of iterations'
pull count
do count
 say 'Hello World' /* or maybe two... */
 end
```

Of course, you can always loop forever.

```
/* Looping Forever */
say 'This program loops until you press CTRL-C'
do forever
 say 'Hello World' /* or three! */
 end
```

Each of these programs repeats the statements between the do and the end the number of times indicated with the do instruction. In the third one, you have to press CTRL-C to break out of the loop.

You can also set the number of iterations in a loop by taking a step-wise approach:

```
/* Stepping Through a Loop */
do count = 1 to 10 by .5
 say 'the square of 'count ' is 'count * count
 end
```

This loop sets an initial condition, an end condition, and a step value. The first time through the loop, count is set to 1. When end is reached, count is incremented by the step value, .5, and the loop is repeated. The loop exits when the count is greater than 10. This loop is similar to a FOR/NEXT loop in BASIC.

Like a FOR/NEXT loop, this style of do loop is useful for stepping through an array. For example:

```
 /* Working with Arrays */
say 'Calculate class average and print the sorted scores'
options prompt 'Enter input:'
say 'Enter number of students in class'
pull number
total = 0
do i = 1 to number by 1
say 'Enter score of student #'i
pull score.i
total = total + score.i
end
do i = 1 to number-1 by 1
 do j = i + 1 to number by 1
    if score.i >= score.j then iterate
    else do
    temp = score.i
    score.i = score.j
    score.j = temp
    end  /* if */
 end  /* inner do */
end  /* outer do */
say 'In order, the scores were:'
```

```
do i = 1 to number by 1
say score.i
end
say 'The average was 'total / number)
```

In this example, you enter the scores into a group of compound symbols that share the score. stem. This sets up an array of variables that you can access using the stem plus a tail that identifies the location of the variable within the array. The do loop is a natural way to step through such an array, letting you access all the variables while using just one name.

The second part of this program is a bubble sort. It will sort any numeric or string array. Every time through the inner loop, it "bubbles" the highest value left in the unsorted part of the array to the current score.i variable. The iterate in the comparison instruction jumps the loop to the end statement (if score.i is greater than or equal to score.j, they are already in the correct order). The bubble sort is not the most efficient sorting algorithm, but it gets the job done.

In many cases, you can't predict how many times you will be going through a loop. The do instruction combines with two others, while and until, to handle these indefinite situations. The while option goes like this:

```
/* Do-While Loop */
response = 'Y'
do while response = 'Y'
say 'enter a number'
pull number
say 'The square of 'number ' is 'number ** 2
say 'Do you wish to square another number (Y/N)?'
pull response
end
```

The until option looks similar.

```
/* Do-Until */
tries = 0
do until flag = 1
say 'What is the product of 6 X 4'
pull response
if response = 24 then flag = 1
else say 'Incorrect, try again.'
tries = tries + 1
end
if tries = 1 then do
```

```
say 'Correct! you got it on the first try!'
exit
end
else if tries = 2 then do
say 'Not bad, you only needed 2 tries.'
exit
end
else do
say 'That''s correct, 6 X 4 equals 24'
say 'You needed 'tries 'tries. Better hit the books.'
end
```

The while instruction keeps the loop running as long as certain conditions remain constant. Until keeps going until a condition reaches a certain level.

Making a Selection

Oftentimes, when you have to make multiple comparisons to discover the value of a variable, using a lot of if-then-else instructions strung together can be difficult to code and understand. ARexx provides the *select* instruction for such occasions. To illustrate, enter this program.

```
/* Menu Selector */
menu:
say 'Do you wish to:'
say 'a) Add two numbers'
say 'c) Cube a number'
say 'm) Multiply two numbers'
say 's) Square a number'
say 'x) Exit
options prompt 'Enter selection:'
pull answer
options prompt 'Enter number:'
select
 when answer = 'A' then do
    pull a
    pull b
    say a 'plus ' b ' equals ' a + b
    end
 when answer = 'C' then do
    pull a
```

```
    say 'The cube of ' a ' is ' a ** 3
    end
  when answer = 'M' then do
    pull a
    pull b
    say a 'times ' b ' equals ' a * b
    end
  when answer = 'S' then do
    pull a
    say 'The square of ' a ' is ' a ** 2
    end
  when answer = 'X' then exit
  otherwise say 'You must enter a menu item'
 end
 signal menu
```

The select instruction tests the when conditions until it finds one that is true. After executing the then instructions, it falls through to the end instruction without making any other tests. If none of the when conditions are true, select executes the otherwise instruction. In the example above, if you don't enter one of the recognized menu choices, you see the message:

```
You must enter a menu item
```

The program will loop until you enter the letter x.

Extending ARexx with the Built-In Functions Library

Although I've glossed over some of the ARexx instructions and skipped over others entirely, the above sections provide a basic working knowledge of the ARexx language. ARexx is a small language, which is part of its appeal. Often, however, you need access to more features than a small language can provide. You can extend the features available to your ARexx program by using the Built-In Functions library, external function libraries, and external hosts.

If you've ever programmed in the C language, you'll be right at home with the Built-In Functions library. Like C, ARexx is a minimal language; it doesn't have hundreds of instructions and keywords. Also like C, it has a standard library of functions that provide essential I/O services and extend the language in various ways. In C, you access the standard language extensions through stdio.h; in ARexx the standard language extensions are in the Built-In Functions library.

The ARexx Built-In Functions library consists of 87 functions concerned with everything from manipulating individual bits to opening files and libraries. You access these functions by calling them by name. (In the discussion below, I capitalize the first letter of function names to differentiate them from internal ARexx instructions.)

For example, one of the functions in the library — Show — lets you find out the name of the available ARexx hosts currently active on your system. These are programs that have ARexx message ports and are able to respond to ARexx commands. You need the name of a port, however, before you can send it a command. To see the port names, you enter:

```
/* Show Ports */
say Show(ports)
```

On my machine, with the ProWrite 3.0 word processor active, this function returns:

```
REXX AREXX ProWrite IPrefs.rendezvous
```

If you need to, you can assign the results of the function called to a variable, and then, using the powerful parsing and string manipulations tools available with ARexx, extract the names of each individual port. You could then test whether the port you need is active.

With Show, you can also test to see if a specific external host is available. For example, to test for the existence of the ProWrite host, you enter

```
test = Show(ports,ProWrite)
```

If the ProWrite port exists, the test variable will contain a 0, a condition you can test. If the port isn't available and you need it to send commands to ProWrite, you can tell the Shell to run ProWrite. More on that later.

The way Show is used in these two examples is representative of how you use the functions in the Built-In Functions library. You call a function by name and include its arguments in an ARexx statement. The functions will either return with some values that you have to handle or it will send a return code that you can check. Because the Built-In Functions library is automatically activated when you start RexxMast, you can consider these functions to be as much a part of the language as say and do.

You can find a short description of the Built-In Functions in Appendix B, and more detailed syntax information in the ARexx portion of your manual. Here, I'll concentrate on the functions you absolutely need to be proficient with

ARexx. These include the file handling functions and functions that communicate with and return information about external hosts. As you use ARexx more and more, you can experiment with its outstanding string-handling functions, as well as with the bit-manipulation and program-debugging functions.

Filing I/O with ARexx

Because ARexx proper doesn't provide any file-handling instructions, they must be provided via Built-In Functions. The primary file-handling functions are Open, Close, ReadCh, ReadLn, EOF, WriteCh, and WriteLn.

Open: Before reading from or writing to a data file from an ARexx program, you must open the file. The arguments you send with the Open functions define the logical name that other file I/O functions will use to access the file, the AmigaDOS pathname, and a character — R, W, or A — that indicates whether you're opening the file to read from it, write over it, or append information to the end of it. For example:

```
return = Open(file_1,'SYS:data','R')
```

tries to open the file Sys:data and makes its contents available for input. The Open function returns a Boolean value that indicates whether it was successful. (0 = failure, 1 = success). Note the use of quote characters around the pathname and the mode indicator. Because the Open function passes these values along to the AmigaDOS file system, you must enclose them in quotes to avoid an Invalid expression error. Knowing when quotes are need and when they are not is one of the most difficult aspects of learning to use ARexx. The safest solution is to enclose all non-numeric arguments in quotes. Occasionally, you may find that you needed two quote marks; at other times, none. Experimentation is the best teacher.

If you try to open a file that doesn't exist for input, Open fails and returns a 0. If you try to open a nonexistent file for output, Open creates the file and returns a 1. Note that you can have a file open for input, close it, and reopen it for writing in the same program.

Close: The Close function takes the logical name of a file as its argument. It closes the file for I/O operations. Although ARexx closes all files when your programs exit, you should close a file when you're done with it so that if the system crashes before the program exits, you won't have errors on your disk.

ReadLn: The ReadLn function reads a line from the indicated logical file and puts it into a variable. For example, if the file Sys:data is open for reading, you

get the next line of data from the file by entering:

```
inputline = ReadLn(file_1)
```

Lines in a file are demarcated by the newline character, which is equivalent to a carriage return. You read a file from the top down. If you are at the end of the file and attempt to read another line, the function returns a null string.

ReadCh: This function reads from an open file the number of characters you supply in the argument. Its other argument is the logical name of the file.

EOF: Unlike some other languages, ARexx doesn't return an error if you attempt a read when you are at the end of a file. It simply returns a null string. If you base branching decisions on whether you've read all the data from a file, however, you want to know when the file is completely read. To learn this, you use the EOF function. For example, assuming the file data_1 is open, you might read from it like this:

```
count = 0
do until EOF(data_1)
count = count + 1
names.count = ReadLn(data_1)
end
Close(data_1)
if count = 0 then say 'File data_1 is empty'
```

This program segment reads the entire contents of the file data_1 into the array names.count. The until condition will skip over the body of the do loop if the file contains no data.

WriteLn: This function writes a literal or a variable to the indicated logical file and appends a carriage return to the end of the output line.

WriteCh: Writes a string or variable to the indicated file. It returns the number of characters it wrote. For example, if the variable name_1 contains a 12-character string,

```
numchar = WriteCh(file_1,name_1)
say numchar
```

will result in

```
12
```

Putting these commands together, you get an idea of how to create and manipulate data files for your ARexx applications.

```
/* Writing and Reading */
Open(file_1,'SYS:data','W')
name.1 = "Katie"
name.2 = "Slim"
name.3 = "Mae"
do i = 1 to 3 by 1
WriteLn(file_1,name.i)
end
Close(file_1)
Open(file_1,'SYS:data','R')
do until EOF(file_1)
say ReadLn(file_1)
end
Close(file_1)
```

Working with External Hosts

Commodore did not include ARexx with Amiga OS 2.0 only because of its wealth of features. The main reason is that a single ARexx program can control the operations of any and all running applications that have ARexx ports, allowing you to mix and match the features of different applications to create super applications.

The primary instructions and functions that provide access to external hosts are address, shell, AddLib, Address, and Show. The first two are ARexx instructions, the last three are functions in the Built-In Function library.

The AddLib Function

Of the commands listed, AddLib is special because it deals with ARexx libraries. It lets you add support libraries to ARexx. Libraries differ from external hosts in that you can access the functions in them without having to specify the library with the address or shell instruction. Once a library is added, you access its functions just as you access the ones in the Built-In Functions library. For example, to add the library Libs:rexxsupport.library to the ARexx system, you enter:

```
AddLib("rexxsupport.library",0,-30,0)
```

Hosts and Ports

Whenever your ARexx program issues a non-ARexx command, a function you created within the program, a Built-In Function, or a function in another library you added with AddLib, ARexx sends the command to the ARexx message port of the current external host program for execution. Only if the host doesn't recognize the command does ARexx report:

```
Function not found
```

To see this in operation, run the following ARexx program:

```
/* Sending a Command to an External Host */
DIR
```

When you run this program, you get a directory listing of the current directory! What's going on here? Dir, as you well know, is an AmigaDOS command. When ARexx encounters Dir in the program, it checks first to see if the command is an internal instruction, then if it is a function defined in the program, then if it's a Built-In Function, and then if it's a function in an added library. After all this, it throws up its (figurative) hands and passes the command along to the current host address. In this case, the current host is REXX, which is synonymous with the Shell from which the ARexx program was invoked. If the host can handle the command, it does. In this case, the Shell does what it always does when it sees Dir, it loads and runs C:Dir.

RexxMast sets REXX as the current host when it starts up. To change the current host to one you choose, you use the address or shell instruction — they are synonymous. For example, to change the host from the current Shell to CygnusEd Professional, a text editor from ASDG, you enter:

```
address 'rexx_ced'
```

This instruction makes 'rexx_ced' the current host and REXX the "previous" host. When entered alone, as in:

```
address
```

the address instruction swaps the current host with the previous one. When you specify yet another host with address, such as

```
address 'ProWrite'
```

the current host is bumped to the "previous" position and the previous one is bumped out of the picture.

To see the name of the current host, you use the Address function. Note that address is an ARexx instruction — it is part of the core language — while Address is a function in the Built-In Functions library.

So see the name of the current host, you enter:

```
say Address()
```

the system responds with the name. Note that Address takes no arguments. To see the names of all the active ARexx hosts, you use the Show function, which was described above.

Putting it All Together

You now know how to find the names of the available external hosts (Show), how to see the name of the current host (Address), and how to make any host the current one (address and shell). The only things you're missing are the commands and arguments to send to the different hosts.

If you've read Chapters 7 through 10, then you have an intimate knowledge of the commands and arguments to send the REXX host. After all, the commands that the REXX host understands are the AmigaDOS commands! What about the other hosts? Programs like Digi-Paint 3, MicroFiche Filer Plus, and ProWrite 3.0? Each ARexx host understands different commands that are documented in their manuals. To use them with ARexx, to combine them into super applications, you must know what commands they support with their individual ARexx interfaces.

Conclusion

You've seen what ARexx does and a bit of how it does it. The next chapter documents an ARexx program that makes use of external hosts. Use it as a guide for your own super applications.

Don't be lulled into thinking that you've seen all that ARexx has to offer. Once you're comfortable with the basic language as I've outlined it here, check out the many options, instructions, and functions that I never touched upon. (You'll be amazed at the number of ways ARexx lets you dissect a string of characters.) This chapter was designed to give you enough background to make sense of Commodore's ARexx documentation. It was an introduction to ARexx. I think you'll find it worth your while to get well-acquainted.

– 12 –

Practical ARexx

The last chapter was concerned with the important ARexx instructions and functions. This chapter presents a complete ARexx application. The point, however, is not to supply you with an application (although that certainly is a by-product of the chapter), but to give you a detailed, annotated example of how ARexx works. By reading about what an ARexx program does line by line, you should be able to see how they are put together.

In the example, I use ARexx instructions, Built-In functions, and commands to two different external hosts. One of the hosts is the Shell, and the other is ProWrite 3.1 from New Horizons software. If you don't own ProWrite 3.1, you simply stop entering the program where I indicate.

The Example Program

The first thing you need to do when writing a program is define what you want it to do. I wanted a program that would create a file that contained the name of every file on any one of my mounted volumes. Also listed, would be the pathname of every file, the date it was created or last modified, and its size in blocks. The list would also be alphabetized. I then wanted the file to be marked with the name of the volume and the date the catalog was created, and then output with page breaks to my printer. The result was Catalog Maker.

Just as I did with the Startup-sequence file in Chapter 10, I've listed Catalog Maker below with a comment for every line. See the section entitled "The Unannotated Program" for a listing without the comments. This latter listing is easier to follow if you're entering it into a text editor.

Because I provide an annotated copy of the program, I don't include comments in the program itself. Normally, you'll want to use comments to remind you of the functions performed by certain sections of your programs. Comments can be a life saver when you try to modify a program you wrote months ago.

The Annotated Listing

In the listings, I've used lowercase for ARexx instructions, variable names, and AmigaDOS filenames; upper/lower case for ARexx functions, and uppercase for Shell and ProWrite commands. Last chapter, I used single quotes as the string delimiters. This chapter, to demonstrate that they are equivalent, I use double quotes as string delimiters.

```
/* Catalog Maker */
```

Every ARexx program must begin with a comment. I use it to name the program.

```
address command
```

This instruction makes REXX — the Shell from which you run the program — the current external host. REXX is the default, so you may wonder why I bother with this instruction. Quite simply, I wanted to be sure that if another ARexx program has altered the current host, my program will still run correctly.

```
ASSIGN >"T:temp VOLS"
```

The program is going to let you choose which volume to catalog. This command lists the currently available volumes to a file. Note that as it is a command to the current external host, I had to put the arguments for the Assign command in parentheses. When it encounters an external command, ARexx evaluates the information after the command, resolves it into a string and then ships it with the command to the host environment. By enclosing the arguments in quotes, you keep ARexx from evaluating the arguments for Assign as ARexx variables or instructions.

Note that if you have deleted the assignment to T: from your Startup-sequence, this program will not work.

```
return = Open(vol_file,"T:temp","R")
```

Because I just created this file, I don't bother checking the return condition. This function opens the file created with the Assign command that contains the names of the mounted volumes.

```
count = 0
```

Sets up a counter. Watch how it is used.

```
do until EOF(vol_file)
```

Vol_file is the logical name I gave T:temp when I opened it. This loop, which reads the volume names from the file, continues until all the names are read. At that point (when end-of-file is True), program control passes to the first line after the end statement. Note that I don't know how many volumes will be mounted, so I can't know how many reads I will need to get all the names.

```
vol.count = ReadLn(vol_file)
```

Now you see that count is actually the subscript for the array vol, which will hold the volume names. When count = 0, this function reads the first line of the file into vol.0.

```
count = count + 1
```

Incrementing count so that subsequent reads go into vol.1, vol.2, and so on.

```
end
```

This instruction ends the do block. Program control now passes to the above do instruction.

```
return = Close(vol_file)
```

Once I've read all the data from T:temp, I can close the file.

```
DELETE ">NIL: T:temp"
```

I don't use the file again, so I might as well get rid of it. The redirection to NIL: keeps unsightly messages from cluttering the output window.

```
count = count - 2
```

I want count to hold the number of volumes available. The current value of count is too high for two reasons. First, the last line in T:temp contains only a linefeed character. This line shouldn't count towards the number of volumes. Second, count gets incremented after the final line is read but before the EOF test. Thus, the actual number of volumes is count - 2.

```
do i = 1 to count by 1
```

The program now loops through the list of volumes, stripping off the "[Mounted]" message that Assign puts there (see Figure 12-1). Note that the loop starts with vol.1, not vol.0. In its output, Assign prints the message

"Volumes:" before actually listing the volumes. Starting the loop at vol.1 chops off this message.

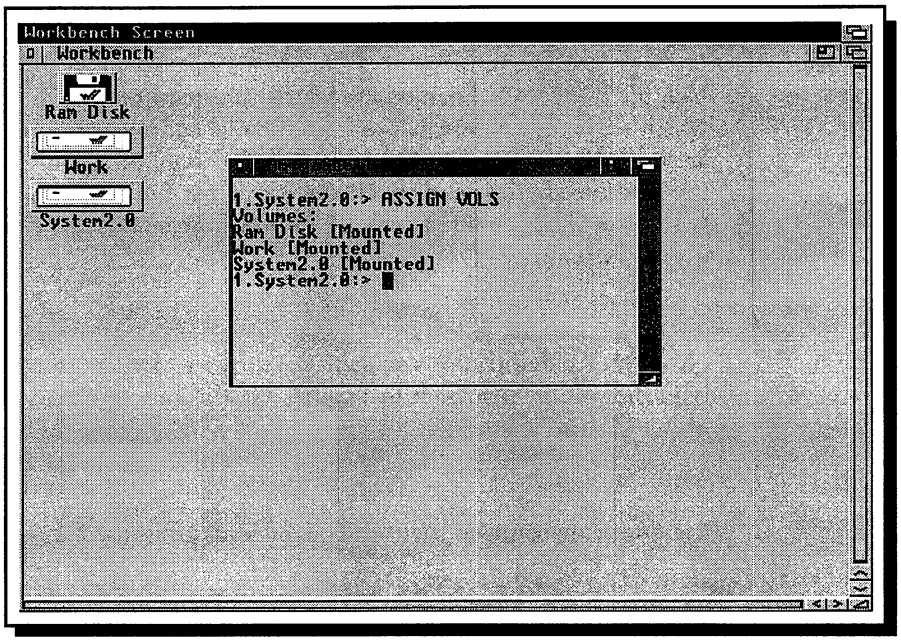


Figure 12-1 Assign VOLUMES

In order to use your menu selection in a command, Catalog Maker must first strip the [Mounted] message from any volume name it appears after.

```
bracket = Pos("["vol.i)
```

The Pos function is one of the many ARexx string-manipulation functions. It reports the position of the character "[" in the string vol.i. The "i" changes each time through the loop, so this check is performed on every volume name.

```
if bracket = 0 then iterate
```

If the volume name doesn't include a bracket (meaning that it is available but not currently mounted), it is already formatted correctly for the next section. This statement skips to the end statement.

```
bracket = bracket - 2
```

Subtracting 2 from the bracket position (1 for the bracket and 1 for the space preceding it), gives you the position of the last character in the volume name.

```
vol.i = Left(vol.i,bracket)
```

This Left function copies only the characters up to the end of the volume name back into the vol array.

```
end
```

Marks the end of the above do loop. Once completed, every item in the vol array contains just a volume name, without any extraneous characters.

```
menu:
```

Provides a label for the next section. If you fail to enter a recognized menu choice, the program displays the menu again.

```
say
```

Prints a blank line to make the output look nicer.

```
say "Indicate the volume you wish to catalog"
```

Prints the above message, getting the user ready for the menu.

```
say
```

Another blank line provided to make the output look better.

```
numchar= C2D("@")
```

The C2D function puts the ASCII value of the character @ into the numeric variable numchar. As it happens, @ is the character just before uppercase A in the table of ASCII codes. I could have looked up the value of @ and used it directly, but why bother when the computer will do the work for you?

```
do i = 1 to count
```

Here is count back again. It still contains the number of volume names in the array vol.

```
say D2C(numchar + i) vol.i
```

First time through the loop, this line prints the letter A followed by the first volume name in vol.i; second time through, it prints B followed by the second name, and so on.

I had to resort to using the ASCII character codes because I had no way of knowing when I wrote the program how many volumes would be available.

The program has to work as well with ten volumes as with one. The D2C is the opposite of C2D; it converts a number into its ASCII character.

```
end
```

Once the program has displayed all the volumes available, the loop ends. Figures 12-2 and 12-3 show the different menus this program creates when there are different numbers of mounted volumes.

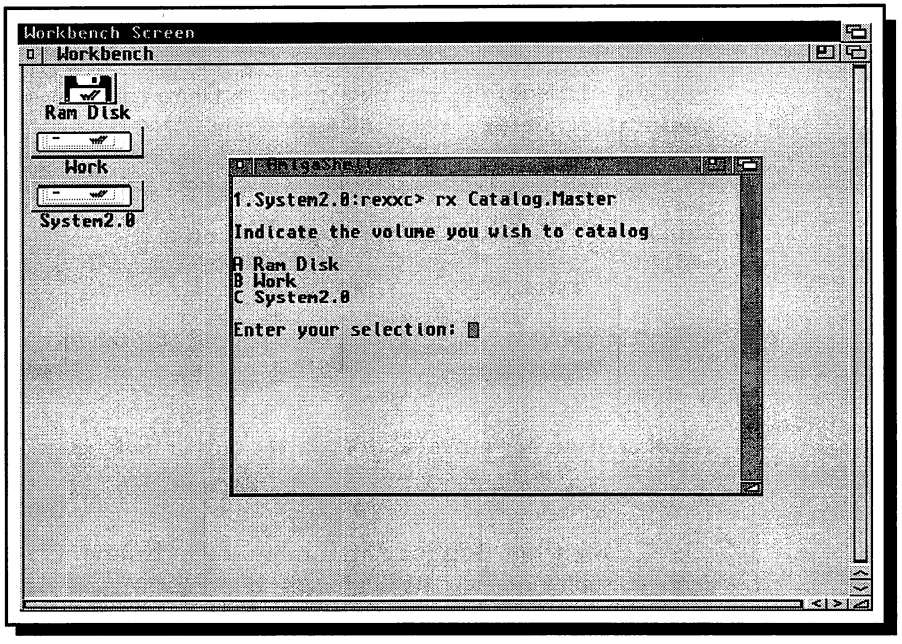


Figure 12-2 Catalog Maker Menu

Using the information from the Assign command, the program puts up a menu showing all the currently available volumes.

```
say
```

Once again, I print a blank line to keep the output lines from crowding each other.

```
options prompt "Enter your selection:"
```

This instruction defines a prompt string to accompany the pull instruction.

```
pull response
```

Prompts the user to enter a selection from the menu created above and places the response into the variable response.

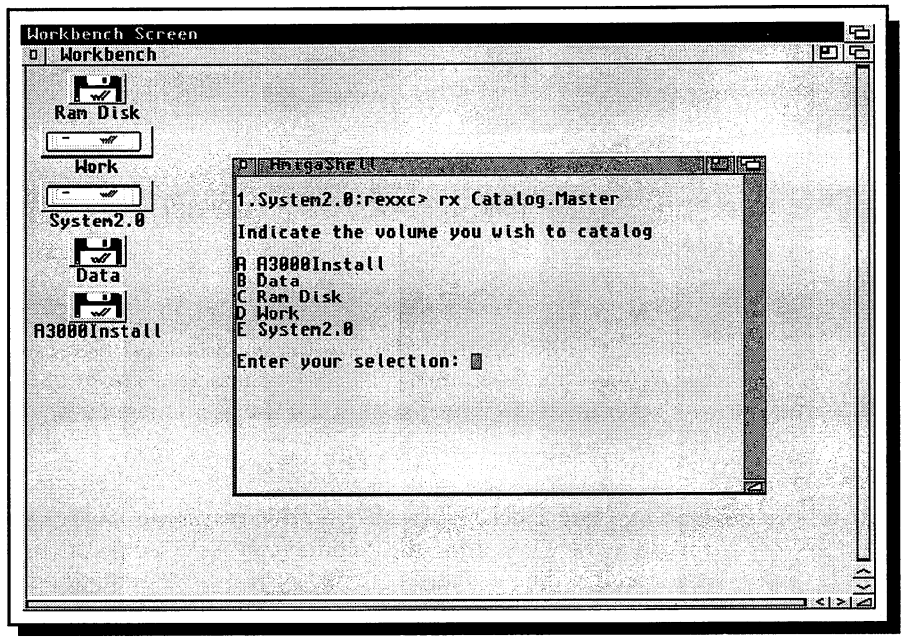


Figure 12-3 Dynamic Menus

No matter how many volumes available on your system, Catalog Maker will display their names in the menu. It determines the number of items to display in the menu automatically.

```
respnum = C2D(response)
```

To check to see if the response is in the range of acceptable values, you first have to convert it into a number.

```
if (respnum <= numchar) | (respnum > numchar +
count) then do
```

Checks to see if the response falls between the ASCII value of the letter A and the value of the last letter in the menu. The test is true if the response is outside the range.

```
say "You must enter a letter from A to "D2C(numchar
+ count)
```

A gentle reminder to the user about what to enter.

```
say
```

Another formatting blank.

```
signal menu
```

Because an invalid response was entered, the program displays the menu again.

```
end
```

Marks the end of the true block. If the letter entered was a valid response, the program continues on the next line.

```
volsub = respnum - numchar
```

Here is how the response relates to the names in the vol array. Subtracting the ASCII value — numchar — of the character @ from the response gives you a 1 if the user entered an A, a 2 if the user entered a B, and so on.

```
volname = """" || vol.volsub || ":"""
```

Don't go into shock. All those quotation marks are easy to explain. What I'm doing here is putting the volume name selected by the user in quotation marks and adding a colon to it. I need to append quotation marks because a volume name can contain a space, and I'm going to be sending the name to AmigaDOS, which doesn't like spaces in names. Let's dissect the first string, which consists of four quotation marks.

The first quotation mark delimits the string, as an initial quote always does. The second one, because it isn't an initial quote yet precedes another, tells ARexx that the next delimiter isn't really a delimiter but that it is actually part of the string. The third quote, because it follows a quote that isn't a delimiter, is treated as a quotation character. The fourth quote, because it is doesn't precede another quote, is treated as a delimiter. (I'll leave it to you to figure out what the ":""" means.)

The upshot of all these quotes is that, if the volume name in vol.volsub is Ram Disk, volname will contain "Ram Disk:". (Remember, || is the concatenation operator.)

```
LIST " >T:cat" volname "DATES BLOCK ALL"
```

The point to the above quotation gymnastics: When ARexx evaluates the arguments for List, it will substitute the volume name for the variable volname.

List puts the information about the files contained on the user-selected volume selected into the file T:cat.

```
return = Open(input_file,"T:cat","R")
```

Opens T:cat for input.

```
return = Open(out_file,"T:list","W")
```

Creates an output file that will hold the formatted catalog information.

```
do until EOF(input_file)
```

This initiates the main processing loop. Here, the file information in T:cat will be converted into the formatted information in T:list. Each line from T:cat will be processed separately.

```
in_line = ReadLn(input_file)
```

If it isn't the end of the file, ReadLn takes the next line from T:cat and puts it into the variable in_line. Note that all files keep an internal pointer, which is automatically incremented whenever you read from a file. The pointer starts at the first line of the file and moves down one line whenever you read a line. You don't have to explicitly move the pointer in the program.

The List ALL command creates different types of output lines (see Figure 12-4), each of which must be handled differently. The following lines test for and handle the different line types.

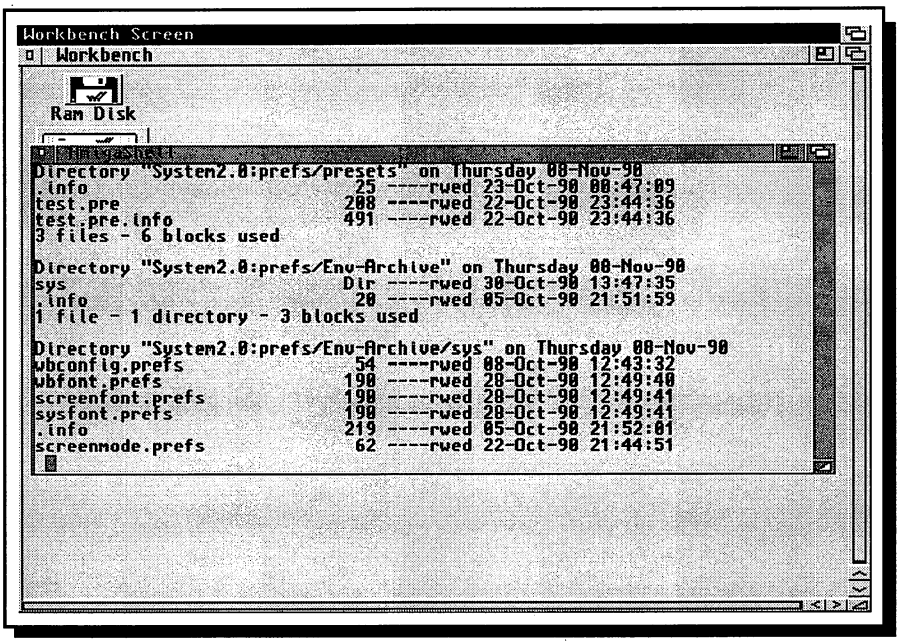


Figure 12-4 List Output

Catalog Maker must handle the different types of lines output by list including blank lines, comment lines, lines that start a new directory, lines that give the number of blocks used by a directory, and, of course, lines with file and directory names.

```
if in_line = "" then iterate
```

Some of the lines output by the List command are blank. By jumping from here to the end instruction, this statement effectively eliminates blank lines from the output file.

```
if Left(in_line,1) = ":" then iterate
```

Comment lines get the same treatment. They are not written to the output file.

```
if Left(in_line,11) = "Directory""" then do
```

Tests to see if the first 11 characters on the line are *"Directory "* as in *Directory "Ram Disk:env" on Thursday 13-Sep-90.* Such message lines are regularly output by a List ALL command, and must be dealt with.

```
len = Length(in_line)
```

If the above test is true, Length returns the number of characters in in_line.

```
in_line = SubStr(in_line,12,len + 1 - 12)
```

This function takes in_line, pulls out all the characters in it except the first 11, and puts them back into in_line. In effect, it deletes the first 11 characters of the current input line.

```
qpos = Pos("""",in_line)
```

Just when you thought you were safe from quotation marks. The Pos function looks for the first occurrence of a quotation character in in_line and returns its position in qpos. With the first 11 characters stripped away, the first quotation is located just after the pathname of the next group of files.

```
curdir = Left(in_line,qpos - 1)
```

Saves the directory pathname of the next group of files in the variable curdir.

```
iterate
```

All you want to do with lines in the form Directory "xxxx" etc. is extract the new current directory. Once you've done that, you want to read from the file again.

```
end
```

Marks the end of the block of statements that handle the directory message lines.

```
if Find(in_line,"blocks used") ~= 0 then iterate
```

Another type of line put out by List ALL that you want to ignore are those that indicate how many blocks are used to store each directory. When this statement encounters such a line, it jumps the program to the end of the file-input loop.

An input line that has failed all the above exception tests is one that contains file information. This is the information we want in the catalog file. The following statements process such lines.

```
timepos = LastPos(" ",in_line)
```

The LastPos function searches a string (such as in_line) *backwards* from the last character in the string — as is the case here — or from a predetermined position in the string, as is the case below. In effect, this statement finds the position of the last space character in the input line. In List command output, the last space always occurs just before the time information.

```
datepos = LastPos(" ",in_line,timepos - 1)
```

The previous statement found the start of the time information by searching for the last space character in the line. This statement starts one character to the left of the last space (timepos - 1) and searches backwards for the next space character which, as a glance at the output of List will tell you, comes just before the data information on a line.

```
propos = LastPos(" ",in_line,datepos - 1)
```

Locates the beginning of the protection information.

```
blockpos = LastPos(" ",in_line,propos - 1)
```

Locates the beginning of the size information.

```
fpos = LastPos(" ",in_line,blockpos - 1)
```

Locates the *end* of the filename information.

```
fname = Left(in_line,fpos -1 )
```

Puts the filename into a separate string.

```
fname = Strip(fname,"T")
```

Strips trailing spaces from the filename.

```
fdate = SubStr(in_line,datepos + 1,timepos - datepos)
```

Extracts the date information from the input line, based upon the position information gathered earlier.

```
block = SubStr(in_line,blockpos + 1,propos - blockpos)
```

Extracts the size information and puts it into block.

```
if block = "Dir" then
```

I want to output information about directories differently from that about normal files, so I perform this test.

```
return = WriteLn(out_file,fname "(Dir)
Path="curdir"  C/M="fdate)
```

In the case of directories, I want the message "(DIR)" printed after the directory name, followed by the path and the date. This statements writes the line to the output file.

```
else
```

If the above conditions are false, the line describes a file.

```
return = WriteLn(out_file,fname " Path="curdir" C/
M="fdate"Size="block)
```

This statement writes the information about files to the output file, including the size of the file in blocks.

```
end
```

When the loop reaches here, control returns to the do instruction, which tests for the end of the input file. When EOF is true, control jumps to the following statement.

```
return = Close(input_file)
```

All the lines have been processed so we might as well close the file.

```
return = Close(out_file)
```

Likewise, there is nothing more to write to the output file.

```
STACK 25000
```

In preparation for the Sort command, I increase the stack size. If you're producing a catalog for a very large hard disk, you may want an even bigger stack.

```
SORT "T:list TO T:sorted.list CASE"
```

This command sorts the lines in the formatted file alphabetically. The CASE argument means that capitalization of the filename has no effect on the sort.

```
STACK 4096
```

With the sort finished, the stack can return to normal, thus freeing more memory for the system.

```
DELETE ">NIL: T:list T:cat"
```

These files have served their purpose and can be discarded.

/* If you don't own ProWrite 3.0 or later you should stop here. Use the AmigaDOS Type command to see the sorted.list file, and Copy to move it to a more permanent location */.

The rest of the program only works if you have ProWrite 3.0 or higher. If your word processor or text editor supports ARexx, you can substitute its formatting commands for the one shown here.

```
if ~Show(Ports,"ProWrite") then do
```

This line tests to see whether ProWrite is currently loaded and in memory. If ProWrite is running, and thus, if its ARexx port is available, the test returns false (the ~ operator negates the normal true result).

```
RUN "Work:ProWrite/ProWrite" /* Note, make sure you
use the location of ProWrite on your disk. This is
where ProWrite is located on my disk. */
```

If ProWrite isn't running, run it now.

```
do until Show(Ports,"ProWrite")
```

Test to see if ProWrite has loaded yet. If not...

```
WAIT
```

...give ProWrite time to load.

```
end
```

This marks the end of the Wait loop.

```
end
```

This marks the end of the test loop. By now, ProWrite is loaded and its port is available.

```
address "ProWrite"
```

Switches REXX from being the current host to being the previous one, and makes ProWrite the host to external commands. Note that port names *are* case sensitive, so you must enclose ProWrite in quotes to keep ARexx from converting the name into all caps.

```
OPEN "T:sorted.list"
```

This command has ProWrite opening the sorted list file.

/* Here you will have to respond to the requester about where to put linefeeds. Just press Return to accept the default. */

Whenever it opens a text file, ProWrite lets you indicate whether the file uses hard or soft carriage returns. Like most Amiga files, sorted.list uses soft returns.

```
TYPE "Catalog of:"
```

This ProWrite command enters the string into the ProWrite document at the current cursor position. When you first open a ProWrite file, the cursor is always at the top of the file, so this string is entered into the top of the file. Note that because we switched the external host with the address instruction, this command is sent to ProWrite and not to AmigaDOS.

```
STYLEBOLD
```

Changes the style of the text input to bold.

```
TYPE vol.volsub
```

Inserts the name of the volume that you cataloged in bold type.

```
STYLEPLAIN
```

Resets the type style to plain text.

```
NEWPARAGRAPH
```

Inserts a hard carriage return into the document. The cursor moves to the first column of the next line.

```
INSERTDATE
```

Enters the system date, in plain text.

```
NEWPARAGRAPH
```

Inserts a hard carriage return.

`NEWPARAGRAPH`

And another, which helps separate the title from the text.

`SELECTALL`

Selects the entire document. The next command will thus work on the entire list.

`COLORBLUE`

Changes any selected text to blue.

`SAVE`

Brings up the ProWrite Save requester, which lets you save the file.

`/* PRINT */`

I left the above command inside a comment. If you want to print the file, delete the comment delimiters. The program will then bring up the ProWrite Print requester.

`QUIT`

Exits ProWrite and, because it is the last command, exits the Catalog Maker program! See Figure 12-5 for a look at how the output looks in ProWrite.

The Unannotated Listing

This is the same listing that appears above. I provide it without commentary to make it easier for you to copy the listing to your text editor.

```
/* Catalog Maker */
address command
ASSIGN ">T:temp VOLS"
return = Open(vol_file,"T:temp","R")
count = 0
do until EOF(vol_file)
vol.count = ReadLn(vol_file)
count = count + 1
end"
return = Close(vol_file)
DELETE ">NIL: T:temp"
count = count - 2
```

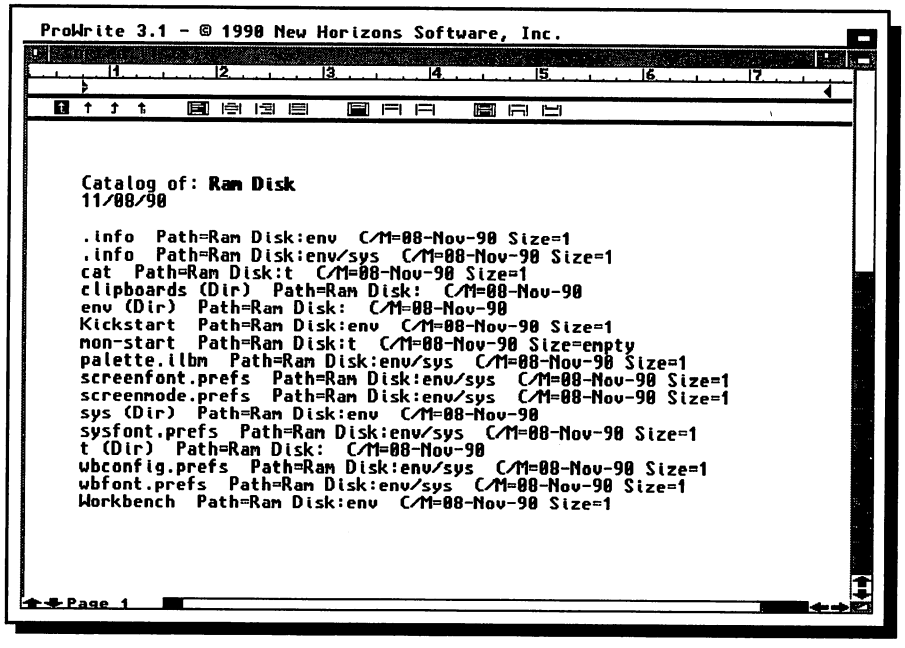


Figure 12-5 Catalog Maker Results

This list was created and formatted automatically by an ARexx program using internal ARexx instructions, Built-In functions, and two external hosts. It shows how you can use ARexx to manipulate other programs.

```
do i = 1 to count by 1
bracket = pos("[",vol.i)
if bracket = 0 then iterate
bracket = bracket - 2
vol.i = Left(vol.i,bracket)
end
menu:
say
say "Indicate the volume you wish to catalog"
say
numchar= C2D("@")
do i = 1 to count
say D2C(numchar + i) vol.i
end
say
options prompt "Enter your selection: "
pull response
```

```
respnum = C2D(response)
if (respnum <= numchar) | (respnum > numchar +
count) then do
say "You must enter a letter from A to "D2C(numchar
+ count)
say
signal menu
end
volsub = respnum - numchar
volname = """" || vol.volsub || ":"""
LIST " >T:cat" volname "DATES BLOCK ALL"
return = Open(input_file,"T:cat","R")
return = Open(out_file,"T:list","W")
do until EOF(input_file)
in_line = ReadLn(input_file)
if in_line = "" then iterate
if Left(in_line,1) = ":" then iterate
if Left(in_line,11) = "Directory """ then do
len = Length(in_line)
in_line = SubStr(in_line,12,len + 1 - 12)
qpos = Pos("""",in_line)
curdir = Left(in_line,qpos - 1)
iterate
end
if Find(in_line,"blocks used") ~= 0 then iterate
timepos = LastPos(" ",in_line)
datepos = LastPos(" ",in_line,timepos - 1)
propos = LastPos(" ",in_line,datepos - 1)
blockpos = LastPos(" ",in_line,propos - 1)
fpos = LastPos(" ",in_line,blockpos - 1)
fname = Left(in_line,fpos -1 )
fname = Strip(fname,"T")
fdate = SubStr(in_line,datepos + 1,timepos -
datepos)
block = SubStr(in_line,blockpos + 1,propos -
blockpos)
if block = "Dir" then
return = WriteLn(out_file,fname "(Dir)
Path="curdir"  C/M="fdate)
else
return = WriteLn(out_file,fname " Path="curdir"  C/
M="fdate"Size="block)
end
```

```
return = Close(input_file)
return = Close(out_file)
STACK 25000
SORT "T:list TO T:sorted.list CASE"
STACK 4096
DELETE ">NIL: T:list T:cat"
/* If you don't own ProWrite 3.0 or later you should
stop here. Use the AmigaDOS Type command to see the
file, and Copy to move it to a more permanent loca-
tion */
if ~Show(Ports,"ProWrite") then do
RUN "Work:Prowrite/ProWrite"
do until Show(Ports,"ProWrite")
WAIT
end
end
address "ProWrite"
OPEN "T:sorted.list"
/* Here you will have to respond to the requester
about where to put linefeeds. Just press Return to
accept the default. */
TYPE "Catalog of: "
STYLEBOLD
TYPE vol.volsub
STYLEPLAIN
NEWPARAGRAPH
INSERTDATE
NEWPARAGRAPH
NEWPARAGRAPH
SELECTALL
COLORBLUE
SAVE
/* PRINT */
QUIT
```

Program Improvements

Catalog Maker merely scratches the surface of the different ways you could format the output. You might want to give the output a more tabular format by inserting the Tab character (ASCII 9) into the output string, or to show file sizes in bytes instead of blocks. You could insert a page break character (ASCII

12) whenever the initial letter of the filename changes. (Of course, this would mean working with the sorted file.) My primary reason for including Catalog Maker was to show how ARexx instructions, functions, and external commands work together; any real use you get out of the program is a bonus.

Conclusion

That wraps up this chapter, the ARexx section, and the book. I hope you'll use Catalog Maker as a starting place for your exploration of ARexx. As you've seen, it is very easy to control different applications from one program. As more Amiga programs support ARexx, you'll have a wider variety of programs to use in your super applications.

I have two major complaints about ARexx: It doesn't let you create Intuition-based programs and it doesn't come with an integrated editor. Just as I was finishing this book, however, I received a preliminary copy of a RXTools from TTR Development Inc., an ARexx programming environment that corrects both shortcomings. I've no doubt that RXTools and products like it will soon make ARexx programming as easy as BASIC programming.

I hope you enjoy your explorations of ARexx and of the Amiga in general. And I hope that this book has helped you on your way.

— A —

AmigaDOS Command Reference

This appendix contains the AmigaDOS commands contained in the C: directory and internal to Kickstart 2.0. I've also included the SetPatch command, which will undoubtedly be added to C: as Commodore patches some of the bugs in Kickstart.

Addbuffers

Function: Add 512-byte memory buffers to a disk device.

Template: DRIVE/A,BUFFERS/A

DRIVE/A is the name of the disk device to which you are adding buffers.

BUFFERS/A is the number of 512-byte buffers to add to the disk. A negative number subtracts buffers, returning the memory to the system.

Alias

Function: Creates an alternate name for a command and lists all current aliases (the default).

Template: NAME,STRING/F

NAME is the label you use on the command line to refer to the STRING/F.

STRING/F is the name and arguments of an AmigaDOS command.

Ask

Function: Get a Y or an N response from the user of a command script. N or a carriage return sets a return code of 0; Y sets a WARN.

Template: PROMPT/A

PROMPT/A is a string written to prompt the user for input.

Assign

Function: Assigns logical names to physical directories and lists previously made assignments (the default).

Template: NAME,TARGET/M,LIST/S,EXISTS/S,DISMOUNT/S, DEFER/S,PATH/S,ADD/S,REMOVE/S,VOLS/S,DIRS/S,DEVICES/S

NAME is the logical name you're giving a physical directory.

TARGET/M are the physical directories to which you are giving a logical name.

LIST/S lists the current logical volumes, directories, and devices. It is the default when Assign is used without arguments.

EXISTS/S sets a WARN if an assigned name already exists.

DISMOUNT/S removes a volume or device from the Assign list.

DEFER/S doesn't confirm the existence of a physical directory until the logical name is used.

PATH/S uses path relative to the root of any volume, as opposed to an absolute pathname, for a TARGET/M.

ADD/S adds the NAME to the assignment list.

REMOVE/S removes a NAME from the assignment list.

VOLS/S lists currently available volumes.

DIRS/S lists currently assigned logical directories.

DEVICES/S lists currently mounted devices.

Avail

Function: Returns the total, available, and in-use amounts of chip and fast RAM available in your system, and the total of the two.

Template: CHIP/S,FAST/S,TOTAL/S,FLUSH

CHIP/S returns the total, in-use, and available, chip RAM in your system.

FAST/S returns the total, in-use, and available fast RAM in your system.

TOTAL/S returns the total, in-use, and available RAM in your system.

FLUSH/S checks the system memory lists to unfragment memory.

Binddrivers

Function: Attaches a device driver in the Expansion drawer to the appropriate expansion device.

Template: None.

Break

Function: Sends an attention flag to a process.

Template: PROCESS/A/N,ALL/S,C/S,D/S,E/S,F/S

PROCESS/A/N is the number of an AmigaDOS process.

ALL/S sends all the flags to the process.

C/S is the CTRL-C flag. It aborts program running in a Shell process.

D/S is the CTRL-D flag. It aborts a Shell command script.

E/S is the CTRL-E flag. Its function is undefined.

F/S is the CTRL-F flag. Its function is undefined.

CD

Function: To list (the default) and change the current directory.

Template: DIR

DIR is the pathname of the new current directory.

ChangeTaskPri

Function: Changes the priority of a process.

Template: PRI=PRIORITY/A/N,PROCESS/K/N

PRI=PRIORITY/A/N is the priority number, from -128 to 127.

PROCESS/K/N is the number of the process with the new priority. The default is the current process.

Copy

Function: To duplicate AmigaDOS files and directories.

Template: FROM/A/M,TO/A,ALL/S,QUIET/S,BUF=BUFFER/K/N, CLONE/S,DATES/S,NOPRO/S,COM/S,NOREQ/S

FROM/A/M lists the files to be copied.

TO/A lists the destination of the copied files.

ALL/S copies the entire AmigaDOS file structure below the copy source.

QUIET/S suppresses the information messages produced during multiple-file copies.

BUF=BUFFER/K/N is the number of memory buffers made available to the Copy command.

CLONE/S copies the file-creation date and the comment of the source file.

DATE/S copies the file-creation date of the source.

NOPRO/S doesn't copy the source file's protection bits.

COM/S copies the source file's comment field.

NOREQ/S suppresses the display of a requester if the destination volume of a copy operation is not mounted.

CPU

Function: Returns information about the state of the CPU (the default) and lets you change that state.

Template: CACHE/S,BURST/S,NOCACHE/S,NOBURST/S, DATACACHE/S,DATABURST/S,NODATACACHE/S, NODATABURST/S,INSTCACHE/S,INSTBURST/S, NOINSTCACHE/S,NOINSTBURST/S,FASTROM/S, NOFASTROM/S,TRAP/S,NOTRAP/S,NOMMUTEST/S, CHECK/K:

CACHE/S enables the instruction and data caches.

BURST/S enables burst mode for instruction and data fetches.

NOCACHE/S disables the instruction and data caches.

NOBURST/S disables burst mode for instruction and data fetches.

DATACACHE/S enables the data cache.

DATABURST/S enables burst mode for data fetches.

NODATACACHE/S disables the data cache.

NODATABURST/S disables burst mode for data fetches.

INSTCACHE/S enables the instruction cache.

INSTBURST/S enables burst mode for instruction fetches.

NOINSTCACHE/S disables the instruction cache.

NOINSTBURST/S disables burst mode for instruction fetches.

FASTROM/S moves the Kickstart image to 32-bit RAM on MMU-equipped machines.

NOFASTROM/S disables the FastROM option.

TRAP/S enables CPU trapping.

NOTRAP/S disables trapping.

NOMMUTEST/S returns 0 if you don't have an MMU, WARN if you do.

CHECK/K lets you check for a specific CPU. It returns a warning if you check for a 68000 on a 32-bit processor machine.

Date

Function: Returns the current settings of the system clock (default), and lets you change the settings.

Template: DAY,DATE,TIME,TO=VER/K

DAY lets you set the date with a day name.

DATE sets the date using DD-MMM-YY format.

TIME sets the time using HH:MM:SS format. SS is optional.

TO=VER/K saves the date to the indicated location.

Delete

Function: Deletes files and directories from disk devices.

Template: FILE/M/A,ALL/S,QUIET/S,FORCE/S

FILE/M/A indicates the files to be deleted.

ALL/S deletes all files below the FILE/M/A argument in the AmigaDOS hierarchy.

QUIET/S suppresses the messages produced during multiple-file deletes.

FORCE/S delete files that don't have the d protection bit set.

Dir

Function: Lists the files and directories in a directory.

Template: DIR,OPT/K,ALL/S,DIRS/S,FILES/S,INTER/S

DIR is the name of directory to be listed. The default is the current directory.

OPT/K specifies an option, A, I, AI, D, and F.

ALL/S lists the contents of the indicated directory and all directories below it in the AmigaDOS hierarchy.

DIRS/S lists only directories, not files.

FILES/S lists only files, not directories.

INTER/S starts an interactive directory listing.

DiskChange

Function: Tells AmigaDOS that you've changed the volume in a disk device.

Template: DEVICE/A

DEVICE/A is the device name of the drive where you changed volumes.

DiskDoctor

Function: Attempts to recover data from damaged disks.

Template: DRIVE/A

DRIVE/A is the device name of the drive holding the damaged volume.

Echo

Function: Prints a string to the console screen.

Template: NOLINE/S,FIRST/K/N,LEN/K/N

NOLINE/S suppresses the carriage return normally appended to an Echo string.

FIRST/K/N sets the first character in the string to be output.

LEN/K/N sets the length of the output string.

Ed

Function: Use the full-screen editor.

Template: FROM/A,SIZE/N,WITH/K,WINDOW/K,
TABS/N,WIDTH=COLS/N,HEIGHT=ROWS/N

FROM/A indicates the name of the file you want to edit.

SIZE/N indicates the size of the text buffer.

WITH/K is the name of a script file of Ed commands that will be executed on the file.

WINDOW/K sets the console device, size, and location of the Ed window.

TABS/S sets the number of space characters in a Tab.

WIDTH=COLS/N sets the number of characters in a line.

HEIGHT=ROWS/N sets the height of the window in lines of text.

Edit

Function: Use the line-oriented editor

Template: FROM/A,TO,WITH/K,VER/K,OPT/K,WIDTH/N,
PREVIOUS/N

FROM/A is the name of the file to be edited.

TO is the name of a destination file.

WITH/K is a macro file of Edit commands executed on the input file.

VER/K designates a file for error messages and verifications.

OPT/K accesses the P and W options, which are equivalent to the PREVIOUS and WIDTH keywords, respectively.

WIDTH/N sets the maximum line length. The default is 120.

PREVIOUS/N sets the number of lines kept in memory. The default is 40.

Else

Function: Indicates the commands in a script that are executed if a condition is false.

Template: None.

EndCLI (see EndShell)

EndIf

Function: Indicates the end of an If block.

Template: None.

EndShell

Function: Shuts down a Shell process.

Template: None.

EndSkip

Function: Stops the search by the Skip command for an indicated label in a script file.

Template: None.

Eval

Function: To perform arithmetic, logical, and bit-manipulation operations.

Template: VALUE1/A,OP,VALUE2,TO,LFORMAT/K

VALUE1/A is the first operand of the operation.

OP indicates the operation to be performed.

VALUE2 is the second operand.

TO indicates the name of the results file.

LFORMAT/K lets you indicate whether you are using hexadecimal, octal, or decimal numbers or ASCII characters.

Execute

Function: Executes a command script file.

Template: None, although it does recognize special characters used in parameter substitution and variable evaluation. See the text.

FailAt

Function: Sets the return value at which a script will abort operation.

Template: RCLIM/N

RCLIM/N is the value of the return code at which a script will abort. The default is 10.

Fault

Function: Returns the error message associated with a particular error number.

Template: /N,/N,/N,/N,/N,/N,/N,/N,/N,/N,

/N is the number of an AmigaDOS error code.

FileNote

Function: Appends or removes a comment to a file or directory.

Template: FILE/A,COMMENT,ALL/S,QUIET/S

FILE/A indicates the files that will get the comment.

COMMENT is the comment string. If absent, the current comment of the indicated files are deleted.

ALL/S appends the comment to all files and directories below FILE/A in the AmigaDOS hierarchy.

QUIET/S suppresses information messages produced by FileNote.

Get

Function: Returns the value of a local environment variable.

Template: NAME/A

NAME/A is the name of the environment variable.

GetEnv

Function: Returns the value of a global environment variable.

Template: NAME/A

NAME/A is the name of the environment variable.

IconX

Function: Executes a command script indicated by a Workbench icon.

Template: None, although its projects support two Tool Types, DELAY and WINDOW. See the text.

If

Function: Test a condition in a command script.

Template: NOT/S,WARN/S,ERROR/S,FAIL/S,,EQ/K,GT/K,GE/K, VAL/S,EXISTS/K

NOT/S is true if the condition is false.

WARN/S is true if the previous return code is greater than or equal to 5.

ERROR/S is true if the previous return code is greater than or equal to 10.

FAIL/S is true if the previous return code is greater than or equal to 20.

EQ/K is true if the two values are equal.

GT/K is true if the first value is greater than the second.

GE/K is true if the first value is greater than or equal to the second.

VAL/S evaluates the values as numbers, not ASCII characters.

EXISTS/K is true if the file or directory exists.

Info

Function: Returns information about all mounted disk devices.

Template: DEVICE

DEVICE returns information about the indicated device.

Install

Function: Makes a volume bootable.

Template: DRIVE/A,NOBOOT/S,CHECK/S,FFS/S

DRIVE/A is the device name where the volume resides.

NOBOOT/S makes a bootable disk unbootable.

CHECK/S checks for a nonstandard boot block. If it finds one, it returns a WARN.

FFS/S uses the Fast File System on the disk.

IPrefs

Function: Launches the IPrefs deamon, which monitors Preferences changes.

Template: None.

Join

Function: Creates a file by joining two or more files together.

Template: FILE/M,AS=TO/K/A

FILE/M are the files to be joined.

AS=TO/K/A is the name of the newly created file. It can't be any of the FILE/M files.

Lab

Function: Sets up a label in a script file for a Skip command.

Template: None.

List

Function: Lists information about file and directories.

Template: DIR/M,P=PAT/K,KEYS/S,DATES/S,NODATES/S,TO/K, SUB/K,SINCE/K,UPTO/K,QUICK/S,BLOCK/S,NOHEAD/S, FILES/S,DIRS/S,LFORMAT/K,ALL/S

DIR/M is the directory to be listed. The default is the current directory.

P=PAT/K indicates the pattern of files to be listed.

KEYS/S lists the key blocks of the files and directories.

DATES/S lists the create/last modified date.

NODATES/S suppresses the date information.

TO/K sends the output of List to a file or device.

SUB/K lists files containing the indicated substring.

SINCE/K lists files created or modified since the indicated day or date.

UPTO/K lists the files created or modified up to the indicated day or date.

QUICK/S lists the file and directory names only.

BLOCK/S reports file sizes in blocks.

NOHEAD/S suppresses headers.

FILES/S lists files only.

DIRS/S lists directories only.

LFORMAT/K embeds the list information in a text string.

ALL/S lists all files and directories below the one indicated in the DIR argument.

Lock

Function: Write protects a volume.

Template: DRIVE/A,ON/S,OFF/S,PASSKEY

DRIVE/A is the volume or device name.

ON/S sets write protection.

OFF/S disables write protection.

PASSKEY is a four-character password you set with ON/S that must be used to disable write protection with OFF/S.

LoadWB

Function: Loads and runs the Workbench interface.

Template: None.

MakeDir

Function: Creates a new directory.

Template: NAME/M

NAME/M is the pathname of the new directory.

MakeLink

Function: Links two files, so that accessing the first actually accesses the second.

Template: FROM/A,TO/A,HARD/S

FROM/A is the name of the link file.

TO/A is the name of the linked file accessed when FROM/A is accessed.

HARD/S is a link between files on the same volume, the only type allowed under the current version of the command.

Mount

Function: To make a device available to AmigaDOS.

Template: DEVICE/A,FROM/K

DEVICE/A is the name of a device.

FROM/K is the name of the mountlist containing information about the device. If not indicated, the mountlist used is Devs:mountlist.

NewCLI (see NewShell)

NewShell

Function: To start a new Shell process.

Template: WINDOW,FROM

WINDOW indicates the console device, height, width, location, and name of the Shell window.

FROM is the name of the Shell-startup file. The default is S:Shell startup.

Path

Function: To view and set the AmigaDOS search path.

Template: PATH/M,ADD/S,SHOW/S,RESET/S,QUIET/S,REMOVE/S

PATH/M are the directories added to the search path.

ADD/S adds the indicated directories to the path. It is the default.

SHOW/S displays the current search path. It is the default if the PATH/M argument isn't used.

RESET/S sets the path to the current directory and C:.

QUIET/S suppresses the search for paths on unmounted disks.

REMOVE/S removes the indicated directory from the path.

Prompt

Function: Sets the command-line prompt.

Template: PROMPT

PROMPT is the new prompt string. It supports two special characters; %N, the number of the current process, and %S, the path of the current directory.

Protect

Function: Modify a file's protection bits.

Template: FILE/A,FLAGS,ADD/S,SUB/S,ALL/S,QUIET/S

FILE/A are the files whose bits you want to modify.

FLAGS are the bits you want to modify.

ADD/S sets the indicated FLAGS.

SUB/S unsets the indicated FLAGS.

ALL/S modifies all files and directories below FILE/A in the hierarchy.

QUIET/S suppresses messages produced by Protect.

Quit

Function: Sets a return code and quits a script.

Template: RC/N

RC/N is the return code of the script.

Relabel

Function: changes the name of a volume.

Template: DRIVE/A,NAME/A

DRIVE/A designates the volume whose name you want changed.

NAME/A is the new name of the volume.

RemRad

Function: Removes a recoverable RAM disk device.

Template: DEVICE,FORCE

DEVICE indicates the name of the RAD: disk you wish to remove.

FORCE removes the disk regardless of its lock status.

Rename

Function: Renames and moves files and directories.

Template: FROM/A/M,TO=AS/A,QUIET/S

FORM/A/M indicates the files and directories you wish to rename.

TO=AS/A is the new name of the files or directories.

QUIET/S suppresses information messages.

Resident

Function: View the list of resident commands and make commands resident.

Template: NAME,FILE,REMOVE/S,ADD/S,REPLACE/S,
PURE=FORCE/S,SYSTEM/S

NAME is the name by which the command will be called in the resident list. If not used, the filename becomes the resident name.

FILE is the pathname of the file made resident.

REMOVE/S removes the indicated name from the resident list.

ADD/S adds the indicated name to the list.

REPLACE/S replaces the indicated name with the new file. This is the default when NAME and FILE are indicated.

PURE=FORCE/S makes the command resident regardless of the setting of the pure bit.

SYSTEM/S means that the name can't be removed from the resident list.

Run

Function: Executes the indicated command in the background.

Template: COMMAND/F

COMMAND/F is the command you want run in the background.

Search

Function: Searches for the indicated string in a file.

Template: FROM/A/M,SEARCH/A,ALL/S,NONUM/S,QUIET/S, QUICK/S,FILE/S,PATTERN/S

FROM/A/M are the files you want searched.

SEARCH/A is the search string.

ALL/S searches all files below FROM/A/M in the file hierarchy.

NONUM/S suppresses the line numbers of matching strings.

QUIET/S keeps quiet about what files are being searched.

QUICK/S uses a compact output format.

FILE/S searches for filenames with the indicated search string, rather than strings within files.

PATTERN/S lets you use pattern matching in the search string.

Set

Function: Display and set local environment variables.

Template: NAME,STRING/F

NAME is the name of the variable. If not indicated, the command displays all the local variables and their values.

STRING/F is the value of the indicated variable.

SetClock

Function: Copy information between the system clock and the battery-backed clock.

Template: LOAD/S,SAVE/S,RESET/S

LOAD/S sets the system clock with the contents of the battery-backed clock.

SAVE/S sets the battery-backed clock from the system clock.

RESET/S resets the battery-backed clock in the event of an error.

SetDate

Function: Sets the time/date stamp of a file.

Template: FILE/A,DATE,TIME,ALL/S

FILE/A are the files you want to stamp.

DATE is the new date stamp, in DD-MMM-YY format.

TIME is the new time stamp, in HH:MM:SS format.

ALL/S uses the new stamp for all files below FILE/A in the AmigaDOS file structure.

SetEnv

Function: Sets global environment variables and displays local ones.

Template: NAME,STRING/F

NAME is the name of a new global variable. If no name is indicated, SetEnv displays the name and values of the *local* environment variables.

STRING/F is the value of the variable.

SetFont

Function: Sets the system default text.

Template: NAME/A,SIZE/A,SCALE/S,PROP/S,ITALIC/S, BOLD/S,UNDERLINE/S

NAME is the typeface of the new font.

SIZE is the size in pixels of the new font.

SCALE/S indicates the font is a scaled version of one of the existing fonts.

PROP/S indicates the font is proportional.

ITALIC/S sets the type style to italic.

BOLD/S sets the type style to boldface.

UNDERLINE/S sets the type style to underline.

SetPatch

Function: Patches the Kickstart image.

Template: None. (Note that this command doesn't appear in the first release of Amiga OS 2 for the Amiga 3000. It will inevitably appear in future releases as Commodore fixes bugs in the OS code.)

Skip

Function: Jump to the indicated label in a command script.

Template: LABEL,BACK/S

LABEL is the label to skip to.

BACK/S searches backwards in the script for the indicated label.

Sort

Function: Orders the lines in a file.

Template: FROM/A,TO/A,COLSTART/K,CASE/S,NUMERIC/S

FROM/A is the name of the file to be sorted.

TO/A is the name of the sorted file.

COLSTART/K indicates the column on which the command is to begin sorting.

CASE/S makes the sort case-insensitive.

NUMERIC/S treats numbers as numeric values rather than ASCII characters.

Stack

Function: Sets the stack size for a process.

Template: SIZE/N

SIZE/N is the new stack size in bytes.

Status

Function: Returns information about all current processes.

Template: PROCESS/N,FULL/S,TCB/S,CLI=ALL/S,COM=COMMAND/K

PROCESS/N lets you indicate which process to display.

FULL/S prints both command and TCB information.

TCB/S prints task-control block information.

CLI=ALL/S produces the default report on all current processes.

COM=COMMAND/K returns the process number of those processes running the indicated command.

Type

Function: Display the contents of a file.

Template: FROM/A/M,TO/K,OPT/K,HEX/S,NUMBER/S

FROM/A/M are the files you want to output.

TO/K indicates the destination of the files. The default is the console window.

OPT/K supports the H and N options.

HEX/S outputs binary files.

NUMBER/S provides a number for every line in the file.

UnAlias

Function: Removes a name from the list of aliases.

Template: NAME

NAME is the alias you want to remove.

UnSet

Function: Removes a local environment variable.

Template: NAME

NAME is the variable you want to remove.

UnSetEnv

Function: Removes a global environment variable.

Template: NAME

NAME is the global variable you want to remove.

Version

Function: Reports the current versions of your Kickstart and Workbench, and sets local variables to those values. Also checks versions of libraries and devices.

Template: NAME,VERSION/N,REVISION/N,UNIT/N

NAME is a library or device whose version you want to check.

VERSION/N is a version number for a library or device. If the version available on the system is greater than or equal to the VERSION/N, the command returns 0; otherwise, it returns WARN.

REVISION/N works the same as VERSION/N, except that it checks revision numbers.

UNIT/N specifies the unit number of a device you want checked.

Wait

Function: Suspends a process for the specified time interval.

Template: /N,SEC=SECS/S,MIN=MINS/S,UNTIL/K

/N is the length of the wait.

SEC=SECS/S indicates that /N is measured in seconds.

MIN=MINS/S indicates that /N is measured in hours.

UNTIL/K indicates that /N is an absolute time in the form HH:MM.

Which

Function: Searches the search path for a particular command and returns its pathname.

Template: FILE/A,NORES/S,RES/S,ALL/S

FILE/A is the command you want found.

NORES/A doesn't include the resident list in the search.

RES/S searches only the resident list.

ALL/S returns all instances of the name in the path, not just the first one.

Why

Function: Attempts to explain why the previous command failed.

Template: None.

– B –

ARexx Quick Reference Guide

Contained in this appendix are the names and functions of the ARexx language instructions, the Built-In Functions, and the rexxsupport.library. See your *Using the System Software* manual for detailed syntax information.

ARexx Instructions

address

Lets you change the current external host. You can toggle between the current and the previous host, or specify a completely new one.

arg

Retrieves argument strings for the program.

break

Exits from a do loop.

call

Invokes either an internal or an external function.

do

Specifies the start of a block of instructions that ends with the corresponding end instruction. Do is also used in conjunction with other instructions to determine how many times the block is executed.

drop

Resets a variable to its uninitialized state.

echo

A synonym for the say instruction.

else

In an if-then-else block, the else instruction begins the block of instructions that are executed if the condition is false.

end

Delineates the end of a do or a select block. When reached in a looping structure, program control returns to the do instruction to test whether an exit condition has been satisfied.

exit

Ends an ARexx program.

if

Tests whether a condition is true or false. If true, the block is executed. If false, an else block may be executed, although this is optional.

interpret

Evaluates an expression and executes the result.

iterate

In effect, jumps to the end instruction in a loop, which then tests the appropriate do condition.

leave

Exits a do loop.

nop

No Operation. It may be needed in an else block.

numeric

Sets the precision and format of numeric data.

options

Sets a fail-at limit, a prompt for the pull instruction, and indicates that a result is expected from an external host.

otherwise

The block of instructions that are executed when all the when conditions in a select block are false.

parse

Extracts substrings from strings. Provides very powerful string handling capabilities.

procedure

Creates a new symbol table, providing for local variables and recursive functions.

pull

Gets input from the keyboard.

push

Puts data on the stack in last-in, first-out order.

queue

Puts data on the stack in first-in, first-out order.

return

Exits a function and returns the result.

say

Writes to the output window.

select

Sets up a block of when instructions which will be tested only until on is found to be true.

shell

Synonymous with the address instruction.

signal

Controls the state of process signal flags and transfers control to a specified label.

when

Similar to if, it tests for a true-false condition within a select block.

The Built-In Functions

Abbrev

Indicates whether one string is an abbreviation of another.

Abs

Returns the absolute value of a number.

AddLib

Adds a function library to the list of libraries available to ARexx.

Address

Returns the name of the current external host.

Arg

Returns the number of arguments passed to the current program.

B2C

Converts binary digits to their character representation.

BitAND

Performs a logical AND on two arguments.

BitChg

Changes the state of a specified bit.

BitClr

Clears the specified bit.

BitComp

Compares two strings for bit differences.

BitOR

Performs a logical OR on two strings.

BitSet

Sets a specified bit in a string.

BitTst

Indicates the state of a specified bit.

BitXOR

Performs an exclusive OR on two strings.

C2B

Converts a character string to its binary equivalent.

C2D

Converts a character string to its decimal equivalent.

C2X

Converts a character string to its hexadecimal equivalent.

Center (Centre)

Centers a string using pad characters.

Close

Closes a file.

Compress

Removes spaces from a string.

Compare

Compares two strings for character differences.

Copies

Creates a string by combining multiple copies of the original.

D2C

Converts a decimal number to its corresponding ASCII character.

D2X

Converts a decimal number to hexadecimal.

Date

Returns the current date.

DataType

Tests whether a string is a valid number.

DelStr

Deletes a substring from a string.

DelWord

Deletes a substring from a string based upon a certain word.

Digits

Returns the digits setting of the numeric instruction.

EOF

Checks for the end of a file.

ErrorText

Returns the message associated with an ARexx error code.

Exists

Tests whether an external file exists.

Export

Copies data into external memory.

Form

Returns the current format setting of the numeric instruction.

Find

Finds a phrase in a string.

FreeSpace

Returns a block of memory back to the interpreter's memory pool.

Fuzz

Returns the fuzz setting of the numeric instruction.

GetClip

Returns a value from the Clip List.

GetSpace

Allocates memory from the interpreter's memory pool.

Hash

Returns the hash value of a string.

Import

Copies data from a specific address.

Index

Searches for the first occurrence of a pattern in a string.

Insert

Inserts one string into another.

LastPos

Searches backwards for the occurrence of a pattern in a string.

Left

Extracts a substring from the left-most characters of a string.

Length

Returns the length of a string.

Lines

Returns the number of lines typed ahead at the keyboard.

Max

Returns the largest of the supplied arguments.

Min

Returns the smallest of the supplied arguments.

Open

Opens an external file.

Overlay

Overlays one string onto another.

Pos

Returns the position of a pattern in a string.

Pragma

Lets a program change its priority or current directory.

Random

Returns a pseudorandom number.

RandU

Returns a uniformly-distributed random number.

ReadCh

Reads a given number of characters from a file.

ReadLn

Reads a line from a file.

RemLib

Removes a library from the ARexx library list.

Reverse

Reverses the characters in a string.

Right

Creates a string from the right-most characters in another string.

Seek

Moves to a new position in a file.

SetClip

Adds a variable name and value to the Clip List.

Show

Shows the contents of the Clip List, the names of currently open files, the contents of the library list, or the names of available external hosts.

Sign

Returns the sign of a number.

SourceLine

Returns the currently executing line of the ARexx program as a string. Also returns the total number of lines in a program.

Space

Sets the number of spaces between words in a string.

Storage

Returns available system memory or stores a string in external memory.

Strip

Deletes leading and trailing spaces from a string.

SubStr

Returns a substring from a string.

SubWord

Returns a substring consisting of whole words.

Symbol

Returns the type of an ARexx symbol.

Time

Returns the current system time.

Trace

Sets the tracing debugging mode.

Translate

Replaces selected characters in a string.

Trim

Removes trailing spaces from a string.

Trunc

Returns the integer portion of a number.

Upper

Converts a string into uppercase.

Value

Returns the value of a variable.

Verify

Returns the position of characters in a string that don't match a specific argument.

Word

Returns a word from a string.

WordIndex

Returns the position of a word in a string.

WordLength

Returns the length of the specified word.

Words

Returns the number of words in a string.

WriteCh

Writes a specified number of characters to a file.

WriteLn

Writes a line to a file.

X2C

Converts a string of hexadecimal digits to their ASCII equivalent.

X2D

Converts hexadecimal to decimal.

XRange

Generates a string of characters that fall between two characters.

The RexxSupport Library

AllocMem

Allocates a block of memory and returns its starting address.

ClosePort

Closes an ARexx message port.

FreeMem

Releases previously allocated memory to the system.

GetArg

Extracts an argument from an ARexx message packet.

GetPkt

Checks a specified message port for available messages.

OpenPort

Opens an ARexx message port.

Reply

Returns a message packet to its sender with a result set.

ShowDir

Returns the contents of a directory.

ShowList

Returns lists of libraries, devices, and ready-and-waiting ports.

StateF

Returns information about a file.

WaitPkt

Waits for a message to be received by a specified port.

– C –

Glossary

AmigaDOS. The part of the Amiga operating system that handles files and devices is called AmigaDOS (Amiga Disk Operating System). The name is also commonly used to refer to the entire Amiga OS.

Amiga OS. The software that controls the operation of the Amiga and manages applications programs is called Amiga OS. It stands for Amiga Operating System.

ARexx. Based upon IBM's mainframe REXX interprocess language, ARexx is a programming language that lets you control many application programs from a single ARexx program.

argument. Instructions understood internally by a command are called arguments. You send arguments to an AmigaDOS command by including them on the command line.

boot. The process of loading the operating system into a computer is called booting. A cold boot occurs when you turn the power on in your computer. A warm boot is caused by a software reset, which you can initiate by pressing the Left-Amiga, Right-Amiga, and Control keys at the same time.

byte. The amount of memory needed to store single alphanumeric characters; a byte consists of eight bits.

chip RAM. Memory used by the custom chips to create your output display or to store sound data is called chip RAM. The CPU can sometimes be shut out of chip RAM for a limited time when both the CPU and the custom chips need to access this memory at the same time. This situation is called bus contention.

command-line interface. A method of controlling a computer where you enter the commands you want performed from the keyboard, and they are interpreted by the program or operating system line by line as you enter them.

command scripts. An AmigaDOS command script is a simple program where the instructions consist of individual AmigaDOS commands. It is also known as a batch file.

CPU. The Central Processor Unit of a computer is the brains of the machine; it executes the software instructions. On the Amiga, the CPU is a microprocessor, a "computer on a chip." The Amiga uses the Motorola 68000 line of microprocessors.

custom chips. The Amiga custom chips perform specialized functions faster than the CPU, thus freeing the CPU to do what it does best. See entries for Agnus, Denise, and Paula in the index.

device. An AmigaDOS device is a software interface between the operating system and the hardware. Such devices recognize a common set of commands that allow applications and the operating system to control the hardware.

directory. A directory is an area on a disk that lists the names and locations of files on the disk.

disk. A flexible or fixed rotating electronic media used to store files.

drawer. The Workbench equivalent of an AmigaDOS directory.

Exec. A library of routines in the Amiga OS that controls such basic functions as memory allocation, multitasking, and interprocess communication.

fast RAM. Memory on the Amiga that is available only to the CPU. If you have fast RAM on your system, AmigaDOS will load programs here to execute them faster. See also chip RAM.

file. A group of related bytes stored on a disk and accessed by name.

format. The process of preparing a disk to hold information in electronic form.

gadget. A graphic picture on the screen that performs a function when you select it. Examples of gadgets are the close, drag, and zoom gadgets attached to a Workbench window.

genlock. The process of synchronizing two video signals so that they can be combined into one.

graphical user interface (GUI). A type of screen display presented to users for control of a computer. A GUI can present information and selections in

graphical form, in contrast to a text-only interface. It commonly features mouse control, multiple windows, pull-down menus, and icons that represent tasks, files, and resources. The Amiga GUI is called Workbench.

HAM. Short for hold-and-modify, this Amiga display mode lets you see 4,096 colors onscreen at the same time.

handler. Software that redefines how AmigaDOS uses a particular hardware resource. For example, the Aux-handler lets AmigaDOS use the serial device for unbuffered two-way communications.

icon. Pictures on the screen that represent operations (such as Copy a file) or objects (such as a file). You use the mouse to point to icons to access the functions the pictures represent.

interprocess communications. A feature of computer operating systems that allows programs (processes) to pass information back and forth among themselves. On the Amiga, the ARexx language can be used to manage interprocess communications, thus accomplishing tasks that require the cooperation of several separate programs.

Intuition. The Amiga Workbench interface is built upon Intuition, a set of routines which controls the creation and operation of windows, menus, and gadgets.

Kickstart. The basic operating system routines of the Amiga are contained in Kickstart. Included are libraries such as Exec, Intuition, graphics, and layers, and device drivers and handlers. While the first release of Kickstart 2.0 was on disk, it will eventually wind up in ROM chips.

library. An Amiga library is a group of software routines that perform related functions. Some libraries are contained in Kickstart, while others reside on disk until a program loads them.

menu. Generally, a list of choices offered by a program. On Workbench, it is a list of related functions accessed by pulling down the menu from the menu bar.

multitasking. The ability of a computer's operating system to run several programs at the same time. Actually, microcomputers multitask by switching among programs so fast that it appears they are executing simultaneously. The two primary forms of multitasking are preemptive, which Amiga OS and other sophisticated operating systems use, and cooperative, used by the Macintosh MultiFinder.

partition. A division on a hard disk that corresponds to one volume.

path. The directories that AmigaDOS searches to find a command you've entered in the Shell.

pathname. The complete description of the location of a file on a disk.

pixel. A picture element. Computer graphics displays are composed of thousands of dots known as pixels.

pointer. The visible analog to the mouse; the pointer lets you select and manipulate objects on the Workbench.

project. Under Workbench, a data file used by an Amiga tool. For example, a paint program creates picture projects.

requester. A simple window created by the Amiga OS or a program that prompts you for input. Normally, you can't continue with a program until you satisfy the input request.

screen. On the Amiga, a screen is a data structure that defines the physical characteristics of your display. It sets the resolution and colors of the display.

Shell. The AmigaDOS command-line interface. On earlier versions of the operating system, it was called the CLI (command-line interface).

Startup-sequence. The command script file in the S: directory of your boot disk that is executed whenever you boot your system.

subdirectory. A name sometimes given to a directory that resides within another directory.

system disk. The disk from which you boot your computer. On a hard disk system, it will be called System2.0. On a floppy-based system, it is Workbench2.0.

tool. Under Workbench, a tool is a program.

trashcan. Under Workbench, a special type of directory available that lets you delete files by dragging them into the trash.

volume. The contents of a disk or disk partition, as opposed to the drive that contains the disk.

window. Rectangular areas on your screen display used by programs to display their output.

Workbench. The program that displays an interface that lets you control the Amiga using a mouse is called Workbench. It is built upon the Intuition library.

– D –

AmigaDOS Error Codes

This appendix lists the AmigaDOS error codes and gives some indication of what causes the indicated errors.

103 — Insufficient free store

There is not enough free memory to run the last command you entered. Try AVAIL FLUSH, or closing down unneeded windows. If your memory list is badly fragmented, you may have to reboot.

104 — Task table full

You probably will never see this error. AmigaDOS used to have a limit of 20 processes. That limit has been eliminated with 2.0.

120 — Argument line invalid or too long

You've entered the arguments for a command incorrectly. Use command history to edit the line and try again. Also note that a command line can't exceed 255 characters.

121 — File is not an object module

You've tried to run a file that isn't a program. Make sure you have the correct name of the executable file. If the file is one you've recently downloaded, be sure that there is no XModem padding appended to the file.

122 — Invalid resident library during load

There is a problem with either of your disk-based libraries or with a resident library. It may indicate an old version of a library. The only way to recover is

to reboot. You may need to copy the Libs: directory from your backup Workbench disk to your working copy.

202 — Object in use

You've tried to access a file that is already being accessed by another program. You'll also get this message if you try to delete a file or directory that has been assigned a logical name.

203 — Object already exists

You've tried to create a directory that already exists. Try another name with MakeDir.

204 — Directory not found

Either you've entered the directory name incorrectly, or it doesn't exist. Try again.

205 — Object not found

Probably the most common AmigaDOS error. It usually indicates that you've entered the name of a command or file incorrectly. It can also mean that you've tried to access a file on a nonexistent directory or unmounted device.

206 — Invalid window description

The window description you've used with NewShell is invalid either because of syntax problems or because the dimensions are out of bounds.

207 — Invalid stream component name

You've entered a file name that is too long or contains a control character.

212 — Object not of required type

You've indicated a directory name where a file is required, or a filename where a directory is required.

213 — Disk not validated

You may have a corrupt disk. If the error persists, you'll have to use DiskDoctor to recover what you can from the disk.

214 — Disk is write-protected

You've tried to write to a write-protected disk. Move the write-protect tab to the enable position or use another disk.

215 — Rename across devices attempted

You can't use Rename to move a file from one disk or disk partition to another. Try the Copy command instead.

216 — Directory not empty

You've tried to delete a directory that has files in it. Delete the files first or use the DELETE ALL option.

218 — Device not mounted

You've tried to access a device that hasn't been added to the system. Most often, this indicates a typing error (such as DFo: for DF0:).

220 — Comment too big

Comments can't be longer than 80 characters.

221 — Disk full

There isn't enough room on the disk to complete the last command. You might try deleting some unneeded files on the disk or use another disk.

222 — File is protected from deletion

The files-deletable bit has been cleared. You can use the Information item from Workbench or Protect from the Shell to set the bit, or use DELETE FORCE to delete the file.

223 — File is protected from writing

This is an error you won't see, because AmigaDOS ignores the w protection bit.

224 — File is protected from reading

Another error that can't occur because AmigaDOS doesn't yet make use of the r protection bit.

225 — Not a DOS disk

This may be caused by inserting an unformatted disk, a corrupt disk, a disk formatted by another operating system, or a copy-protected disk. If the disk is corrupt, you'll have to call on DiskDoctor.

226 — No disk in drive

Put a disk in the drive or use a drive that has a disk.

Index

— D —

— J-K —

— L —

— Q —

— R —

— T —

— U —

— V —

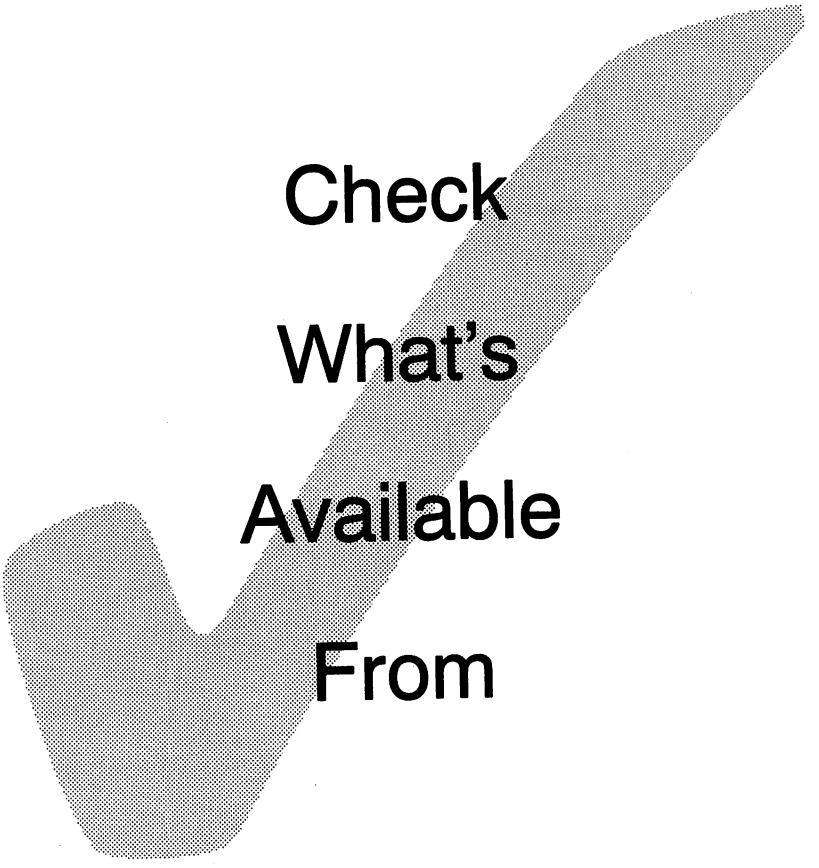
— W —

— X-Z —

About The Author

Bob Ryan is currently a Technical Editor at *BYTE* magazine, where he works on the State of the Art section. He is the author of the magazine's recent cover story on the Amiga 3000 computer. Formerly, Bob was the first Technical Editor of *AmigaWorld* magazine, where he was immersed in all aspects of Amiga technology. He holds a BA degree from Clark University.

Bob resides in New Ipswich, NH with his wife, daughter, dog, and two cats. When not delving into the innards of the Amiga, he enjoys his family and friends, reads history, plays a little golf, and roots real hard for the Boston Red Sox.

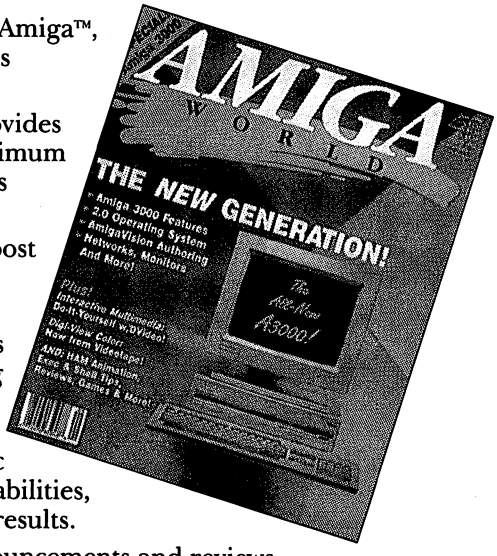

Check
What's
Available
From

AMIGA WORLD

STEP INTO THE WORLD OF AMIGA...

The Pathway To Your Imagination

For a computer as extraordinary as the Amiga™, you need a magazine that can match its excellence, *AmigaWorld.*



AmigaWorld is the only magazine which provides you with ideas and information to get maximum performance from the Amiga's tremendous power and versatility.

Each issue gives you valuable insights to boost your productivity and enhance your creativity.

Whether you choose the Amiga as a serious business tool for its speed and multitasking capabilities...or for its superb graphics, drawing, color (over 4,000 colors), and animation...or for its state-of-the-art music and speech...or for its scientific and CAD abilities, *AmigaWorld* can help you achieve superior results.

With its timely news features, product announcements and reviews, useful operating tips and stunning graphics, *AmigaWorld* is as dynamic as the market it covers.

Don't wait! Become a subscriber and save 58% off the cover price. Return the coupon, or for immediate service, call toll-free **1-800-258-5473.**

Save 58% ☑ *YES!*

I want to discover the full potential of this powerful machine. And save 58% off the cover price. Enter my one year (12 issues) subscription to *AmigaWorld* for the special price of $19.97. If I'm not satisfied at any time, I will receive a full refund—no questions asked.

❑ Payment enclosed ❑ Bill me

Name_____

Address_____

City_____ State_____ Zip_____

Canada & Mexico $29.97, Foreign Surface $49.97, Foreign Airmail $84.97 (U.S. Funds drawn on U.S. bank). Prepayment required on foreign orders. All rates are 1 year only. Please allow 6—8 weeks for delivery.

AmigaWorld PO Box 58804, Boulder, CO 80322-8804

5DGA

THE *AMIGAWORLD* TOOL CHEST

Quality Software At An Affordable Price!

Here's some of what you'll get with

VOLUME 2, ISSUE 1:

★ Circe. Battle your computer to take over the planet Circe.

★ Colors. A programmer's color reference utility for assigning RGB values to printer and screen output.

★ EasyFile. This powerful database manager is suitable for both home and small business.

★ Sprite Editor. Create animated sprites by editing up to 100 frames at once.

★ Sound Effects. Digitized sounds of a car in trouble, driving, a creaky rocker, sawing wood and a frightened crowd.

★ Batchman. This handy utility allows you execute CLI programs, batch files and ARexx scripts with the simple click of a gadget.

TOOL CHEST

is brought to you by the same top-quality editors who publish *AmigaWorld*. Whatever your skill level may be—or whatever model of Amiga you own—you'll be thrilled with how Tool Chest can help make breakthrough computing inexpensive, easy and fun.

Every dual-disk issue of Tool Chest is loaded with entertaining games, elaborate animation, exquisite 3D, useful utilities, original clip art, and wild sound effects to help you maximize the value of your Amiga computer.

EXCLUSIVE OFFER

Subscribe to Tool Chest, save $20.00 off the single copy price, and receive these special subscriber benefits:

★ *AmigaWorld* subscription/ renewal discounts

★ Discount off AmiExpo admissions

★ Full Money Back Guarantee. If ever you are not completely satisfied with the *AmigaWorld* Tool Chest, simply return it for a full refund.

From graphics to animation, from programming to productivity, you can do it faster and easier with the Tool Chest. If you want the work you are producing on your Amiga to be the very best, subscribe to Tool Chest today.

☐ **YES!** I want to save $20.00 and receive my special subscriber benefits. Enter my one year (6 dual-disk issues) subscription for only $69.95.

❑ Send me the following *AmigaWorld* Tool Chest Edition(s) for only $14.95 each: ❑ Volume 2, Issue 1 (see above)
❑ Specify other _____

Name _____

Address _____

City _____ State _____ Zip ____

❑ Check/Money Order enclosed
❑ Charge my ❑ MasterCard ❑ American Express ❑ Visa ❑ Discover

Acct. # _____ Exp. Date _____

Signature _____

Foreign orders, add $2.50 for air mail delivery. Foreign subscriptions are $94.95 postpaid. Payment required in U.S. Funds drawn on U.S. bank. Note that some animations require 1MB of memory.

AmigaWorld Tool Chest • 80 Elm Street • Peterborough, NH 03458

Or call 800-343-0728 for immediate service IDGBKS

The *AmigaWorld* Video Library

It's like having a professional computer consultant at your side, 24 hours a day!

Animation Video Volume I

When the Editors of *AmigaWorld* canvassed the Amiga community looking for the best in Amiga animation, the response was overwhelming! Submissions poured in from Amiga master artists and super-talented readers. The result is a dramatic video featuring dozens of world-class animators. This video is quickly becoming a collector's item! Approximately 48 minutes in length. Available in NTSC.

Desktop Video

Don't miss out while others get the inside angle on: Pre-production, production and post-production; Home and studio settings; Selecting video equipment and accessories; Edit-free shooting with your camera; Recording from the Amiga onto video tape; Selecting a Genlock and how to get the most from it; Tips and Tricks on how to improve your video skills; Adding special effects; *And lots more!*

The Musical Amiga

Your Amiga has exquisite sound and music capabilities. Learn all the details of getting started with music. . . what you need to begin and playing music with existing software. We'll teach you digitizing and audio sampling and even give you an introduction to MIDI (Musical Instrument Digital Interface). This video is the best instructional tool for any Amiga owner interested in learning how to create music with their Amiga!

Amiga Graphics (Volume I)

There's no easier, faster or better way to learn how to create your own Amiga masterpieces! This hour-long video can teach you all you need to know about how to get started in graphics. . .paint programs. . . elements of design. . .creating an image. . .and lots more! Plus, three extensive sessions on FONTS, CLIP ART, and even DIGITIZING!

Getting Started With Your Amiga

This comprehensive, easy-to-follow video is packed with valuable information. Learn how to assemble your Amiga. . .how to use the Workbench. . .add a digitizer or genlock. . .how to use system utilities. . .and much, much more! Best of all, GETTING STARTED WITH YOUR AMIGA is there whenever you need a quick refresher on any aspect of Amiga computing!

AMIGA WORLD

Fill out coupon and mail to:
Video Library, PO Box 802, 80 Elm Street, Peterborough, NH 03458

or call 1-800-343-0728

☑ **YES!** Please send me the tapes I've selected below!

Qty.	Description	Price	Total
	Getting Started With Your Amiga	**$29.95**	
	Amiga Graphics (Volume I)	**$29.95**	
	Desktop Video	**$29.95**	
	The Musical Amiga	**$29.95**	
	Animation Video (Volume I)	**$19.95**	
		Subtotal	
		Shipping/Handling *	
		Total Enclosed	

Name _____

Address _____

City _____ State _____ Zip _____

Payment method: [ ] Check

Charge my: [ ] MC [ ] VISA [ ] American Express [ ] Discover

Card # _____ Exp. Date _____

Signature _____

* Shipping/Handling Per Order:
1 Tape = $2.95
2 or more Tapes = $5.00

Available in VHS only. Make checks payable to *AmigaWorld* Video Library. Please allow 4–6 weeks for delivery. Foreign orders add $7.50 for air mail orders. Payment must be made in US funds which can be drawn on a US bank. PAL is available if specified. Add an additional $10.00 for PAL version plus $7.50 air mail delivery. IDG/Peterborough, publisher of *AmigaWorld*, is the licensed North American distributor of all above-mentioned videos. Amiga is a registered trademark of the Commodoore Business Machines, Inc.

International Data Group (IDG), International Environment Group, Inc. (IEG), an affiliate of IDG based in Peterborough, N.H., and IDG Books Worldwide, Inc., an affiliate of IDG based in San Mateo, CA, are committed to directing the power of business and industry toward improving the environment.

▲▼▲

This book was printed on recycled paper, and can be recycled.

IDG Books Worldwide Registration Card -
AmigaWorld Official AmigaDOS Companion

Fill this out to hear about updates to this book and new information about other IDG Books Worldwide products. Thank you!

Name _____

Company/Title _____

Address _____

City/State/Zip _____

What is the single most important reason you bought this book? _____

_____

Where did you buy this book?

- ❒ Bookstore (Name _____)
- ❒ Electronics/Software Store (Name _____)
- ❒ Advertisement (If magazine, which? _____)
- ❒ Mail Order
- ❒ Other:

How did you hear about this book?

- ❒ Book review in: _____
- ❒ Advertisement in: _____
- ❒ Catalog
- ❒ Found in store
- ❒ Other: _____

How many computer books do you purchase a year?

- ❒ 1
- ❒ 2-5
- ❒ 6-10
- ❒ More than 10

How would you rate the overall content of this book?

- ❒ Very good
- ❒ Good
- ❒ Satisfactory
- ❒ Poor

Why? _____

What do you like best about the book? _____

What do you like least? _____

What other topics/products would you like to see added to future editions of this book?

_____

Please give us any additional comments. _____

_____

Thank you for your help.

❒ I liked this book! By checking this box, I give you permission to use my name and quote me in future IDG Books Worldwide promotional materials.

FOLD HERE

Place stamp here

IDG Books Worldwide, Inc.
155 Bovet Road, Ste. 730
San Mateo, CA 94402

Attn: Reader Response

AMIGA WORLD

Official AmigaDOS 2 Companion

CERTIFIED IDG TECHNICAL EDIT

Get into the new Amiga operating system—and get more out of it!

This is the most comprehensive, authoritative guide you can find to using AmigaDOS and the new Amiga operating system. You get step-by-step instructions, hundreds of tips and screen shots, and definitive references to using Amiga OS 2 via the workbench, AmigaDOS & the Shell, and ARexx!

Inside, find expert help in these important areas:

- The Amiga OS, including the Workbench, Shell, and ARexx
- The Workbench GUI—windows, screens, menus, gadgets, requesters, the mouse pointer
- Detailed information on Workbench menus
- The 13 Preferences editors and their functions
- Workbench tools—system programs & the Commodities Exchange
- The Extras 2 disk—new programs and hard disk utilities
- AmigaDOS—handling disks, files, devices via the shell
- Configuring AmigaDOS and getting information out of AmigaDOS
- Manipulating files with: Copy, Delete, MakeDir, Rename, and more
- AmigaDOS command scripts
- Getting up to speed with ARexx macro language
- A fully annotated ARexx program that you can customize
- And more!

Plus, command references, valuable glossary, and error code summary for solving problems more easily.

Computer Book Shelving Category

Amiga/AmigaDOS/Operating System

$24.95 U.S./$33.95 Canada/£22.95 U.K.

PRINTED ON RECYCLED PAPER

Designed by Owens/Lutter. Art by Jim Sachs, created on an Amiga with DeluxePaint 3.
Amiga is a registered trademark of Commodore-Amiga Inc.

Book Level:

Beginning to Intermediate

About the Author:

Bob Ryan is currently Technical Editor at *BYTE* magazine, working on the State of the Art section. Formerly, Bob was the first Technical Editor of *AmigaWorld* magazine.

IDG BOOKS

IDG Books Worldwide
San Mateo, CA 94402
An International Data Group Company

ISBN 1-878058-09-6

9 781878 058096 52495