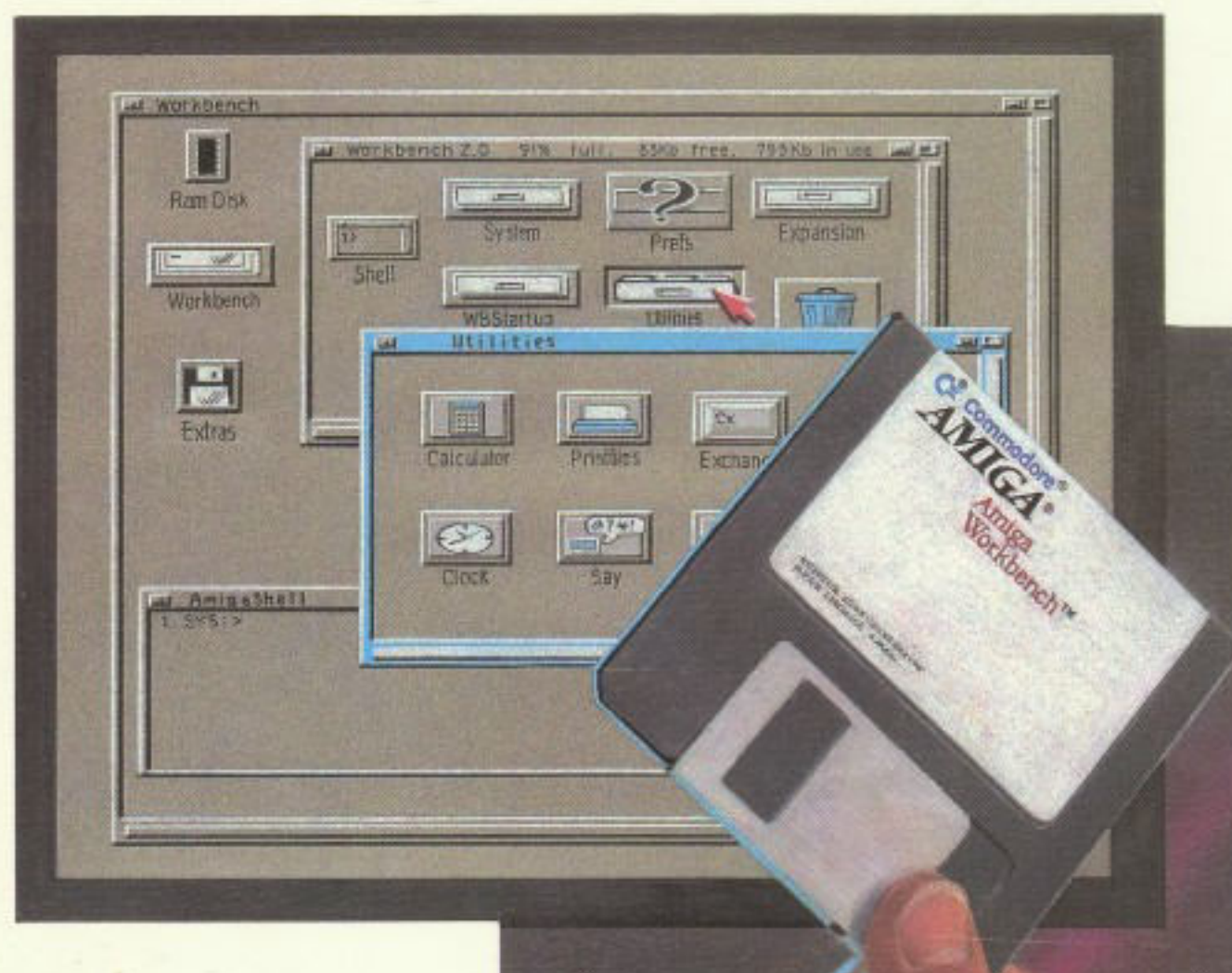


WORKBENCH™ • AMIGADOS™ • EDITORS • UTILITIES

Using The System Software



 **Commodore®**
AMIGA®

WORKBENCH™ • AMIGADOS™ • EDITORS • UTILITIES

Using The System Software

 **Commodore®**
AMIGA®

COPYRIGHT

© 1990, 1991, Commodore-Amiga, Inc. All Rights Reserved. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form, without prior consent, in writing, from Commodore-Amiga, Inc.

The material set forth in Chapters 7, 8, and 9 is adapted from *The AmigaDOS Manual*, 2nd Edition, Copyright © 1987 by Commodore-Amiga, Inc. used by permission of Bantam Books. All Rights Reserved.

The Times Roman, Helvetica Medium, and Courier fonts included in the Fonts directory of the Extras2.0 disk are Copyright © 1985, 1987 Adobe Systems, Inc.

The BRU program is Copyright © 1988, Engineering Software Tools. The HDBackup program is Copyright © 1989, 1990 Enhancer Software Technologies, Inc.

If this product is being acquired for or on behalf of the United States of America, its agencies and/or instrumentalities, it is provided with RESTRICTED RIGHTS, and all use, duplication, or disclosure with respect to the included software and documentation is subject to the restrictions set forth in subdivision (b) (3) (ii) of The Rights in Technical Data and Computer Software clause at 252.227-7013 of the DOD FAR. Unless otherwise indicated, the manufacturer/integrator is Commodore Business Machines, Inc., 1200 Wilson Drive, West Chester, PA 19380.

DISCLAIMER

This information is provided "as is" without representation or warranty of any kind, either express or implied, including without limitation, any representations or endorsements regarding the use of, the results of, or performance of the information, its appropriateness, accuracy, reliability, or currentness, the entire risk as to the use of this information is assumed by the user.

In no event will Commodore, its affiliated companies, nor its employees, be liable for any damages, direct, indirect, incidental or consequential, resulting from any defect in the information, even if Commodore has been advised of the possibility of such damages.

This disclaimer shall supersede any verbal or written statement to the contrary.

TRADEMARKS

Amiga is a registered trademark of Commodore-Amiga, Inc.; The Amiga check mark, Amiga 500, Amiga 2000, A3000, AmigaDOS, Amiga Workbench and Amiga Kickstart are trademarks of Commodore-Amiga, Inc.; Commodore, the Commodore logo and CBM are registered trademarks of Commodore Electronics Ltd. BRU is a trademark of Engineering Software Tools.

ColorMaster is a trademark of CalComp.; Diablo and Xerox are registered trademarks of Xerox Corporation; Epson is a registered trademark of Epson America, Inc.; IBM and Proprinter XL are registered trademarks of International Business Machines Corp.; Imagewriter is a trademark of Apple Computer, Inc.; LaserJet, LaserJet PLUS, and PaintJet are trademarks of Hewlett-Packard Company; Microsoft and MS-DOS are registered trademarks of Microsoft Corp.; NEC and Pinwriter are registered trademarks of NEC Information Systems; Okidata is a registered trademark of Okidata, a division of Oki America, Inc.; Okimate 20 is a trademark of Okidata, a division of Oki America, Inc.; Tektronix is a registered trademark of Tektronix, Inc.; UNIX is a registered trademark of AT&T; VT100 is a registered trademark of Digital Equipment Corp.

This document may also contain references to other trademarks which are believed to belong to the sources associated therewith.

About Your Documentation

Three standard documents are included with your Amiga:

1. *Quick Connect – How to Set Up the Amiga*

This booklet, which folds out into a large poster, shows you how to connect your Amiga equipment and any peripherals, such as a monitor or printer. You can hang this poster in a convenient spot while you follow the set up instructions.

2. *Introducing the Amiga*

This manual describes the major components and features of your Amiga, and tells you what to expect when you first turn on the computer. The manual also introduces you to the **Workbench**, which is the software interface that lets you interact with your computer through graphic symbols appearing on the screen. The expansion capabilities of the Amiga are also covered.

3. *Using the System Software*

This manual, which explains how to use the software packaged with your Amiga, is divided into three main parts: **Workbench 2.0**, **AmigaDOS**, and **AREXX Programming Language**. The manual begins with a tutorial aimed at the first-time user. Subsequent chapters build on this base and cover the Workbench and AmigaDOS in detail. A comprehensive guide to AREXX completes the manual.

Here's what's in each chapter and appendix:

Chapter 1, Tutorial, takes you step-by-step through the elementary tasks involved in using your Amiga.

Chapter 2, Basic Operations, expands on the tutorial to provide a more detailed explanation of how the Amiga works.

Chapter 3, Preferences, tells you how to properly set your Amiga to work with monitors, printers, and other peripheral devices and how to customize your Workbench screen (e.g., by changing colors and type fonts).

Chapter 4, *The Workbench Programs*, explains all the programs on the Workbench disk, like Say, which lets you enter text for the Amiga to speak.

Chapter 5, *The Extras Programs*, explains all the programs on the Extras disk, like GraphicDump, which lets you print out screen images.

Chapter 6, *Using a Hard Disk*, explains how to use the Amiga's hard disk to store and retrieve files and programs. The chapter also provides step-by-step instructions for backing up the hard disk and adding new hard disks to the system.

Chapter 7, *Using AmigaDOS*, introduces you to the terms and concepts behind AmigaDOS. The discussion includes a description of the Shell, a keyboard-based interface that lets you run programs and perform basic operations through typed commands, and explanations of some of the most basic AmigaDOS commands.

Chapter 8, *AmigaDOS Reference*, lists and describes all the AmigaDOS commands and error messages.

Chapter 9, *Editors*, explains the three text editors included with the Amiga: ED, EDIT, and MEMACS.

Chapter 10, *AREXX*, fully describes a new and powerful programming language.

Appendix A, *Troubleshooting*, contains a list of possible problems and suggested solutions.

Appendix B, *Printers*, lists many of the printers that can be used with the Amiga, as well as the standard printer escape sequences.

Appendix C, *Backing Up Your Hard Disk with BRU*, describes BRU, a sophisticated backup-and-restore program for Shell users. The appendix also tells you how to customize BRU's defaults, discusses BRU's help screen, gives individual explanations of BRU commands, and lists sample commands to create archives and restore files.

The *Glossary* defines important terms used throughout the manual.

How To Use This Documentation

If you have never used an Amiga before, read *Quick Connect* and *Introducing the Amiga*. Then read Chapters 1 through 3 in this manual. Finally, as you need or want more detailed information on the Amiga's operations and specific features, read the remaining chapters.

If you have used an Amiga before, you should first read *Quick Connect* and *Introducing the Amiga*. In *Using the System Software*, you may want to skip the tutorial, but you should read the other chapters as needed to learn what's new about Workbench and AmigaDOS.

A Word About Graphic Symbols

The following graphic symbols appear in the margins:



This symbol draws attention to instructions that must be read carefully in order to avoid damage to your system.



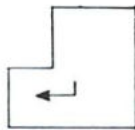
This hard disk symbol directs you to information important to hard disk users.

A Word About Keys

All references to alphabetical keys are shown in uppercase letters. Unless otherwise specified, do not press Shift. If the instructions read "Press Q," simply press the Q key. If an uppercase letter must be used, it will be specified in the instructions.

Non-alphanumeric keys are shown as they appear on the keycap (Ctrl, Esc, Del, Alt, Help).

The Amiga keys are referenced by their position: left Amiga (**A**) and right Amiga (**A**). Several keys on the keyboard have arrows on the keycaps. The list below shows the keycap and the name used for that key:



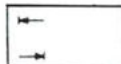
Return



Backspace



Shift



Tab

The group of arrow keys to the left of the numeric keypad are the cursor keys. The keys are referenced by the direction of their arrow: up cursor, down cursor, left cursor, and right cursor.

At times, you need to press a sequence of keys. In these instances, the keys are separated by a hyphen and shown in the order they should be pressed, such as Ctrl-O. This means you must press, and hold, the Ctrl key, then press the O.

A Word About Type Styles

Bold words

Words that appear in bold are defined in the glossary.

Screen Output

Words that appear in this style can be one of two things: input you must type at the keyboard or output that is shown on the Amiga screen. This style is especially prevalent in the AmigaDOS section of the manual.

Contents

Workbench 2.0

1. Tutorial

Getting Started.....	1-2
Using the Mouse	1-6
The Selection Button	1-7
The Menu Button	1-9
Using Menus.....	1-11
Ghosted Menu Items	1-13
Working with Windows	1-14
The Title Bar.....	1-16
The Depth Gadget.....	1-17
The Zoom Gadget	1-19
The Sizing Gadget	1-20
The Scroll Gadgets	1-21
The Close Gadget	1-25
Making Backup Copies of Disks.....	1-27
Using One Disk Drive	1-29
Using Two Disk Drives	1-33
Renaming Your Backup Disks.....	1-38
Rebooting the Amiga	1-40
Using Application Software.....	1-42
Formatting a Disk	1-44
Organizing Information on a Disk.....	1-47
Paths.....	1-51
Naming Files.....	1-54

2. Basic Operations

The Workbench System	2-2
Mouse Techniques.....	2-5
The Selection Button	2-8
Selecting	2-8
Selecting Multiple Icons	2-9
Double-Clicking.....	2-11

Dragging	2-11
Dragging an Icon	2-11
Dragging a Window	2-12
Dragging a Screen	2-13
The Menu Button	2-15
Using Menus	2-15
Cancelling	2-19
Requesters	2-20
Action Requester	2-21
Text Requester	2-22
File Requester	2-24
The Workbench Screen	2-26
Title Bar	2-26
Workbench Window	2-27
Moving the Workbench Screen	2-27
Windows	2-28
Title Bar	2-30
Zoom Gadget	2-31
Depth Gadget	2-31
Scroll Gadgets	2-33
Sizing Gadget	2-35
Close Gadget	2-37
Additional Gadgets	2-38
Action Gadgets	2-39
Check Box	2-40
Cycle Gadget	2-41
Radio Button	2-42
Scroll Gadget	2-43
Selection Gadget	2-45
Slider Gadget	2-46
Text Gadget	2-47
Icons	2-49

The Workbench Menu	2-50
Backdrop	2-51
Execute Command	2-52
Redraw All	2-54
Update All	2-54
Last Message	2-55
Version	2-55
Quit	2-56
The Window Menu	2-58
New Drawer	2-59
Open Parent	2-60
Close	2-61
Update	2-61
Select Contents	2-62
Clean Up	2-63
Snapshot	2-63
Show	2-64
View By	2-65
The Icons Menu	2-67
Open	2-67
Copy	2-68
Copying by Dragging	2-73
Rename	2-75
Information	2-76
Snapshot	2-79
Unsnapshot	2-80
Leave Out	2-80
Put Away	2-81
Delete	2-81
Format Disk	2-82
Empty Trash	2-86
The Tools Menu	2-87

3. Preferences

The Prefs Drawer	3-2
Action Gadgets	3-4
The Time Editor	3-5
Save/Use/Cancel	3-6
The Input Editor	3-7
Mouse Speed	3-8
Acceleration	3-9
Double-Click	3-9
Key Repeat Delay	3-10
Key Repeat Rate	3-11
Key Repeat Test	3-11
Save/Use/Cancel	3-11
The Palette Editor	3-12
Save/Use/Cancel	3-14
The Workbench Pattern Editor	3-15
Color Selection Gadget	3-19
Magnified View Box	3-20
Actual Size Box	3-20
Presets Gadget	3-20
Pattern	3-20
Test	3-21
Clear	3-21
Undo	3-21
Save/Use/Cancel	3-22
The Pointer Editor	3-22
Test	3-24
Clear	3-25
Set Point	3-25
Reset Color	3-25
Save/Use/Cancel	3-25
The Font Editor	3-26
Text Radio Buttons	3-27

Font Gadget.....	3-28
Text/Field.....	3-29
Save/Use/Cancel	3-30
Types of Displays.....	3-30
Hires	3-33
SuperHires.....	3-34
Productivity	3-35
A2024.....	3-36
The Screen Mode Editor.....	3-37
Choose Display Mode.....	3-37
Properties of the Selected Mode.....	3-38
Screen Sizes.....	3-39
Width	3-40
Height.....	3-40
Colors.....	3-41
AutoScroll	3-41
Save/Use/Cancel	3-41
The Overscan Editor.....	3-42
Edit Text Overscan	3-44
Edit Standard Overscan	3-46
Screen Sizes.....	3-47
Save/Use/Cancel	3-48
The Printer Editor	3-49
Paper Length.....	3-52
Left Margin	3-52
Right Margin.....	3-53
Printer Port	3-53
Paper Type.....	3-53
Paper Size	3-54
Print Pitch.....	3-55
Print Spacing.....	3-55
Print Quality.....	3-56
Save/Use/Cancel	3-56

The Printer Graphics Editor	3-57
Color Correct	3-57
Smoothing	3-59
Left Offset/No. Inches	3-60
Center Picture	3-60
Dithering	3-60
Scaling	3-63
Image	3-64
Aspect	3-64
Shade	3-65
Threshold	3-65
Limits/Type	3-66
Width	3-69
Height	3-69
Density	3-69
Save/Use/Cancel	3-69
The Serial Editor	3-70
Baud Rate	3-70
Input Buffer Size	3-71
Handshaking	3-71
Parity	3-72
Bits/Char	3-73
Stop Bits	3-73
Save/Use/Cancel	3-73
The IControl Editor	3-74
Verify Timeout	3-74
Command Keys	3-75
Mouse Screen Drag	3-75
Coercion	3-76
Screen Menu Snap	3-77
Text Gadget Filter	3-77
Save/Use/Cancel	3-78
The Workbench Configuration Editor	3-79
Save/Use/Cancel	3-80

The Editor Menus and Presets Drawer	3-80
The Project Menu	3-81
The Edit Menu	3-82
The Options Menu	3-82
Using the Presets Drawer	3-83

4. The Workbench Programs

Tool Types	4-2
Adding a Tool Type	4-2
Deleting a Tool Type	4-4
Changing a Tool Type	4-4
The System Drawer	4-5
AddMonitor	4-7
BindMonitor	4-10
DiskCopy	4-11
FixFonts	4-11
Format	4-12
NoFastMem	4-12
SetMap	4-13
Adding a Tool Type	4-16
The Utilities Drawer	4-18
Clock	4-19
The Type Menu	4-20
The Mode Menu	4-20
The Seconds Menu	4-20
The Date Menu	4-21
The Alarm Menu	4-21
Tool Types	4-23
Display	4-23
Tool Types	4-25
Exchange	4-27
More	4-29
Say	4-32
Tool Types	4-34
The WBStartup Drawer	4-35
Tool Types	4-36

5. The Extras Programs

The MonitorStore Drawer	5-2
The Tools Drawer	5-3
Calculator	5-4
CMD	5-7
Colors	5-8
Using Colors	5-10
GraphicDump	5-12
IconEdit	5-13
Color Selection Gadget	5-14
Magnified View Box	5-14
Freehand Gadget	5-14
Continuous Freehand Gadget	5-15
Circle Gadget	5-15
Box Gadget	5-16
Line Gadget	5-16
Fill Gadget	5-17
Undo	5-17
Clear	5-17
Normal/Selected Radio Buttons	5-17
Arrows	5-18
IconEdit Menus	5-19
Project	5-19
Type	5-20
Highlight	5-21
Images	5-22
Misc	5-23
InitPrinter	5-24
KeyShow	5-24
PrintFiles	5-26
Tool Types	5-27
The Commodities Drawer	5-27
AutoPoint	5-30

Blanker	5-30
Tool Types	5-31
FKey	5-32
Tool Types	5-34
IHelp	5-34
Tool Types	5-35
NoCapsLock	5-36

6. Using a Hard Disk

About Your Hard Disk	6-1
Hard Disk Partitions	6-3
Copying Software to Your Hard Disk	6-4
Troubleshooting	6-7
Adding an ASSIGN Statement to your User-Startup File	6-8
Example ASSIGN Statement	6-11
Backing Up Your Hard Disk	6-14
HDBackup	6-15
Using HDBackup for the First Time	6-15
Creating a Full Backup	6-16
Creating an Incremental Backup	6-27
Selected Files and Selected Size Display	6-29
Include and Exclude Gadgets	6-29
File Selection Gadgets	6-30
Smaller Log File Option	6-35
File Compression Option	6-36
Checking Differences	6-38
Inspecting a Backup	6-40
Restoring Files	6-42
Tool Types	6-45

HDDToolBox	6-49
The Hard Drive Preparation, Partitioning and Formatting Screen	6-50
Partitioning	6-53
Adjusting the Size of a Partition	6-56
Sliding a Partition within the Partitioning Bar	6-56
Adding a New Partition	6-57
Renaming a Partition	6-57
Deleting a Partition	6-58
Using HDDToolbox's Default Setup for the Drive	6-58
Saving and Formatting Your New Partitions	6-59
Advanced Options with Partitioning	6-60
Preparing a New Hard Disk	6-63
Low Level Formatting	6-72
Locating Bad Blocks	6-75
Adding a Bad Block to the Bad Block List	6-78
Changing the Drive Type	6-80
Editing a Drive Type or Defining a New Drive Type	6-82
Modifying File Systems	6-86
File System Maintenance	6-89
To Add a New File System	6-91
To Delete a File System	6-92
To Update an Existing File System	6-93

AmigaDOS

7. Using AmigaDOS

Introduction to AmigaDOS	7-1
The File System	7-2
Devices	7-3
Peripheral Devices	7-4
Directories	7-5

Files.....	7-6
.info Files.....	7-7
Paths.....	7-8
Naming Files and Directories.....	7-9
Basic AmigaDOS Commands.....	7-11
Types of AmigaDOS Commands.....	7-11
The Shell.....	7-13
Getting Information About Disks.....	7-15
Creating a New Directory.....	7-18
Changing the Current Directory.....	7-19
Changing the Search Path.....	7-22
Working with Files.....	7-23
Working with Disks.....	7-25
Setting the Clock.....	7-27
Opening/Closing Shell Windows.....	7-28
Special AmigaDOS Characters.....	7-29
Command Line Characters.....	7-29
Pattern Matching.....	7-31
Redirection.....	7-33
Features of the Shell.....	7-34
Editing.....	7-35
Copying and Pasting.....	7-37
Customizing the Window.....	7-38
Closing the Shell.....	7-41
The Shell-startup File.....	7-41
Using Aliases.....	7-41
Changing the Prompt.....	7-43
Using Escape Sequences.....	7-43
Running Programs.....	7-46
Scripts.....	7-49
Condition Flags.....	7-51
Environment Variables.....	7-52
The Startup-Sequence.....	7-55
Editing the Startup-Sequence.....	7-62
Common Additions to the Startup Files.....	7-64
To automatically open a Shell window:.....	7-65

To set up additional paths and logical device names:	7-65
To make additional commands resident:	7-65
For Single Floppy Disk Systems	7-66
Making Commands Resident	7-67
Using Assign's PATH Option	7-68
Making Room on Your Workbench Disk	7-69
The Ram Disk	7-72
Advantages for Floppy Disk Systems	7-72
The Recoverable Ram Disk	7-74
For a 1MB Amiga	7-75
For Amigas with more than 2MB	7-77
Other Workbench Directories	7-78
The S: Directory	7-78
ED-Startup	7-79
HDBackup.Config	7-79
SPat,DPat	7-79
PCD	7-80
The DEVS: Directory	7-80
MountList	7-81
Keymaps	7-84
Printers	7-85
The L: Directory	7-85
Aux-Handler	7-86
Pipe-Handler	7-86
Port-Handler	7-87
Speak-Handler	7-87
The FONTS: Directory	7-89
The LIBS: Directory	7-90

8. AmigaDOS Reference

Command Conventions	8-1
Format	8-3
Template	8-4
Command Specifications	8-7
Error Messages	8-131

9. Editors

ED	9-1
New Features of ED	9-2
Starting ED	9-3
Using ED	9-5
Immediate Commands	9-6
Moving the Cursor	9-7
Inserting Text	9-8
Deleting Text	9-9
Changing Case	9-9
Extended Commands	9-10
Program Control	9-11
Cursor Control	9-14
Altering Text	9-15
Block Control	9-16
Searching and Exchanging	9-18
Repeating Commands	9-20
Customizing ED	9-21
AREXX Support	9-25
MEmacs	9-28
Starting MEmacs	9-28
Using MEmacs	9-29
Menu Commands	9-32
The Project Menu	9-33
The Edit Menu	9-38
The Window Menu	9-44
The Move Menu	9-46
The Line Menu	9-48
The Word Menu	9-50
The Search Menu	9-52
The Extras Menu	9-54
Commands Not In Menus	9-59
Customizing MEmacs	9-61

EDIT	9-63
Starting EDIT	9-64
Using EDIT	9-65
The Current Line	9-65
Prompts	9-66
Output Processing	9-67
Commands	9-68
Arguments	9-69
Multiple Commands	9-72
EDIT Commands	9-73
Selecting the Current Line	9-73
Editing the Current Line	9-75
Inserting and Deleting Lines	9-77
Splitting and Joining Lines	9-82
Renumbering Lines	9-83
Verifying Lines	9-84
Inspecting the Source File	9-85
Making Global Changes	9-86
Changing Command, Input and Output Files	9-87
Ending EDIT	9-91

AREXX

10. AREXX Programming Language

Introduction	10-1
Who is AREXX For?	10-2
What Do You Have to Know to Use AREXX?	10-3
Chapter Organization	10-3
AREXX on the Amiga	10-4
Multitasking	10-5
Interprocess Communication	10-5
Interprocess Communication and Ports	10-5
Multitasking and Interprocess Communication Together	10-6
What is AREXX?	10-7

Starting AREXX on the Amiga.....	10-7
Automatically	10-8
Manually	10-8
Setting AREXX to Start Automatically	10-9
Displaying Output	10-13
Naming AREXX Programs	10-13
Where Do AREXX Programs Go?	10-14
AREXX System Files	10-14
SYS:LIBS Directory	10-14
SYSTEM: Directory	10-14
SYS:Rexxc Directory	10-15
SYS:Rexx (An Assign Designated to Store AREXX Programs).....	10-15
Language Features.....	10-15
Program Examples.....	10-15
Instructions	10-18
Review and Additional Notes	10-24
Elements of AREXX	10-26
Format	10-27
Tokens	10-27
Clauses	10-31
Clause Classification	10-34
Clause Continuation	10-34
Expressions	10-35
Symbol Resolution	10-36
Order of Evaluation.....	10-36
Numbers and Numeric Precision	10-37
Boolean Values.....	10-38
Numeric Precision	10-38
Operators.....	10-39
Stems and Compound Symbols	10-44
The Execution Environment.....	10-46
The External Environment	10-47
The Internal Environment	10-47
Input and Output	10-49
Resource Tracking.....	10-49
Instructions	10-50

Commands	10-74
Command Clauses	10-74
The Host Address	10-75
The Command Interface	10-78
Using Commands in Macro Programs	10-79
Using AREXX with Command Shells	10-81
Command Inhibition	10-82
Functions	10-82
The Library List	10-83
Syntax and Search Order	10-84
Search Order	10-85
The Clip List	10-89
The Built-In Function Library	10-90
Built-in Functions	10-91
RexxSupport.Library Functions	10-129
Tracing and Interrupts	10-134
Tracing Options	10-135
Display Formatting	10-136
The Global Tracing Console	10-137
Tracing Output	10-138
Command Inhibition	10-139
Interactive Tracing	10-140
Error Processing	10-141
Failure Level for Commands	10-142
The External Tracing Flags	10-142
Interrupts	10-143
Parsing and Templates	10-146
Template Structure	10-146
Template Objects	10-148
The Scanning Process	10-149
Templates in Action	10-150
Pattern Parsing	10-151
Positional Markers	10-152
Multiple Templates	10-153
Additional Notes	10-154
REXXC Directory	10-154
Error Messages	10-157

Appendices

A. Troubleshooting

B. Printers

Printer Drivers	B-1
Escape Sequences	B-22

C. Backing Up Your Hard Disk with BRU

Brutab	C-2
The Components of the Brutab File	C-3
Setting Environment Variables for BRU	C-6
BRU Command Lines	C-7
Interrupting BRU While it is Operating	C-9
Using BRU — A Tutorial	C-10
Estimating the Size and Creating a	
Backup of the Current Directory	C-10
Backing Up Your Entire Hard Disk	C-12
Backing Up Only a Few Files or Directories	C-13
Combining Modes	C-14
Adding Control and Selection Options	C-14
Restoring Files to Your Hard Disk	C-17
BRU Argument Reference Section	C-20
Modes	C-20
Creating an Archive	C-20
Differences in Size	C-20
Estimating the Size	C-21
Give Information on Archive Header	C-22
Print the BRU Help Screen	C-23
Inspecting the Archive	C-23
Listing the Table of Contents	C-24
Extraction	C-25
Control Options	C-26
Amiga Specific Flags	C-26
Set Archive Buffer Size	C-26
Telling BRU to Run Without User	
Intervention	C-26

Use Path as the Archive File	C-27
Fast Mode	C-28
Interaction Option	C-28
Labeling an Archive	C-29
Use nbits for Compression	C-29
Pass Over Archive Files by Reading	C-29
Pathname Handling and Expansion	C-30
Exclude Remotely Mounted Files	C-30
Specify the Size of Archive Media	C-30
Turn on Sparse File Options	C-31
Setting the Verbosity Level	C-31
Asking BRU to Wait for Confirmation	C-32
Use LZW File Compression	C-33
File Selection Options	C-33
Selecting Files by Date	C-33
Unconditional File Type Extraction	C-35
BRU with UNIX	C-36
Control Options	C-36
Control String	C-36
Do Not Reset Access Time	C-36
Change the Owner of Extracted File	C-36
Double Buffer	C-37
Interaction Option	C-37
Ignore Unresolved Links	C-37
Limit Directory Expansions to Same Mounted Filesystem	C-38
Select Files Owned by User	C-38
Selection Options	C-39
Unconditional File Type Extraction	C-39
BRU Error Messages	C-39
Messages Starting with Filename	C-40
Messages Starting with Warning	C-46
Other Messages	C-50

Chapter 1. Tutorial

This chapter introduces you to the Amiga® system and the **Workbench™** software by telling you how to:

- turn on your machine
- use the mouse
- choose options from menus
- make backup copies of your floppy disks
- prepare floppy disks for data storage
- organize your files on a disk

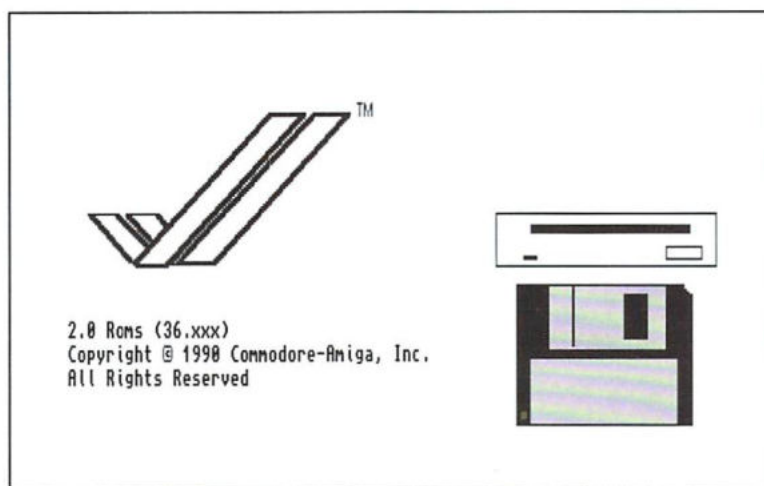
When you finish this tutorial you will know enough about your Amiga to begin running application software, such as spreadsheets, word processors, desktop publishing or graphics programs.

Don't worry if it seems that some ideas are not explained in full detail. Chapter 2, "Basic Operations," provides more complete explanations.

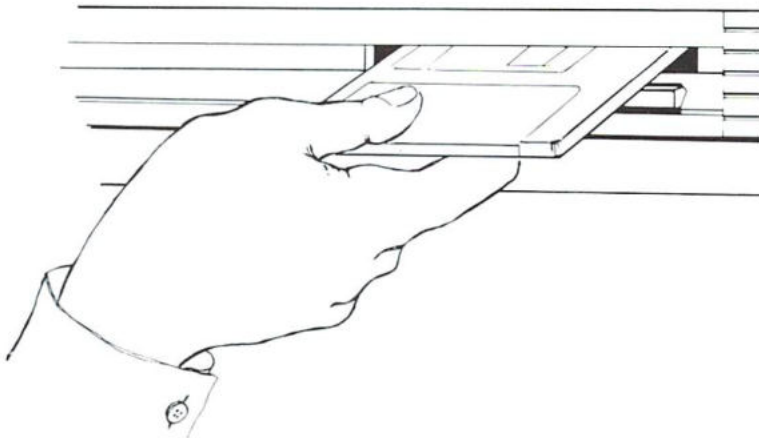
Getting Started

Turn on the power to your Amiga as shown in your *Introducing the Amiga* manual. The power light on the front of your machine signals that the power is on. You must also turn on the power to your monitor. Consult the documentation that came with your monitor for instructions.

If you have a floppy disk system, an animated screen appears showing a floppy disk being inserted into a disk drive. This is your signal to insert the **Workbench2.0** disk into the disk drive.



If you have an autobooting hard disk in your Amiga, you will not see the animated screen. The system software is already installed on your hard disk, so you do not need to insert the Workbench2.0 disk into the disk drive. If you do see the animated screen, you have a problem with your hard disk and should consult your dealer or authorized Commodore service center.



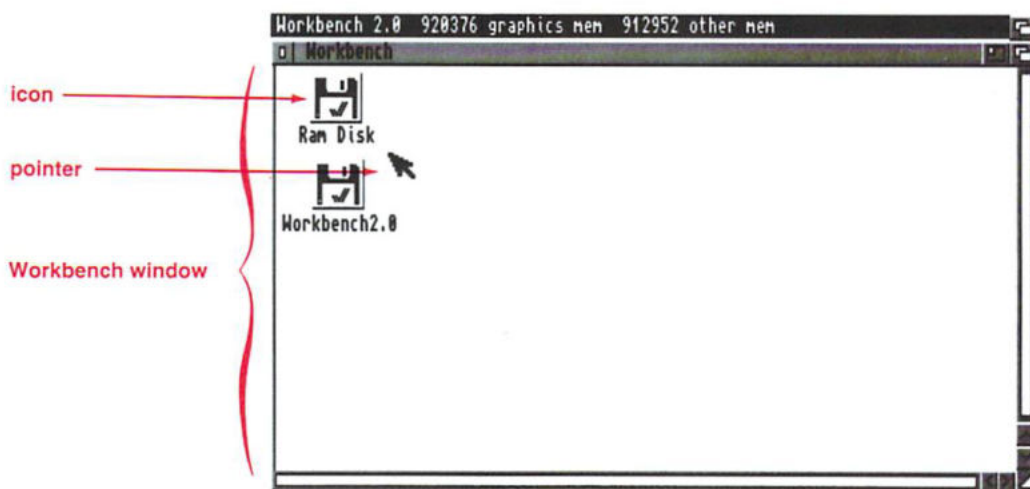
1. *Insert the Workbench2.0 disk into the disk drive, metal end first, with the labeled side facing up .*

The process of turning on your computer and loading information from a disk containing the Amiga operating system is known as **booting** your computer—from the old expression “pulling yourself up by your bootstraps.” It refers to the process of reading the information needed to start the system from a storage device, such as a floppy or hard disk, into the computer’s memory.

The disk drive light comes on as the Amiga starts reading the information on the disk. Never remove a floppy disk from the drive while the disk drive light is on.



The first thing that you'll see on your monitor is some introductory information, such as copyright notices and the name and version numbers of the software. After a few seconds, the introductory screen is replaced by the **Workbench screen**. It is through this screen that you access the programs on the Workbench2.0 disk.



A **window** (an area on the screen that accepts and displays information) fills most of the screen. This window, called the Workbench window, contains two **icons**. Icons are images that represent various items, such as disk or files.



If you have a hard disk system, you may have different or additional icons in this window, but you can still follow this tutorial. (Consult your *Introducing the Amiga* manual for an accurate depiction of the screen.) When this manual refers to the Workbench2.0 disk icon, just substitute the System2.0 hard disk icon. Some of the screens shown in this manual may vary from what actually appears on your screen, but the procedures remain the same.

The icon with the word Workbench2.0 underneath represents the actual Workbench2.0 disk that is in your Amiga's disk drive. Whenever you insert a disk into the drive, an icon representing that disk appears in this window.

The icon labeled **Ram Disk** represents a section of the Amiga's internal memory that is available for temporary data storage.

The Ram Disk acts like a floppy disk in that you can store information in it and retrieve information from it. However, the computer can access information on the Ram Disk much faster than information on a floppy disk.

The contents of the Ram Disk are erased if the Amiga is turned off.

There is also a **pointer** on the screen. The pointer is controlled by the mouse. By positioning the pointer over objects on the screen and pressing a mouse button, you can start programs, exit programs, copy disks, rearrange your screen, and much more.

RAM stands for Random Access Memory.

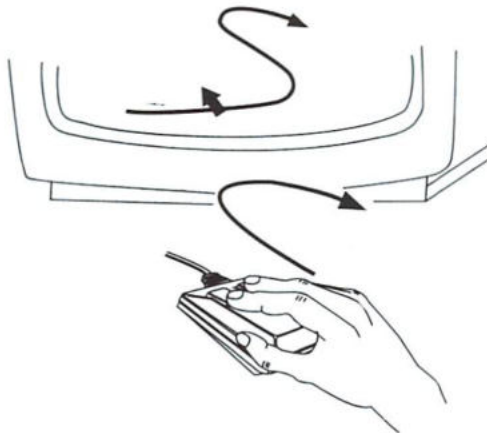


You can change the shape and color of the pointer with the Pointer editor in the Prefs drawer (explained in Chapter 3).

Using the Mouse

The mouse controls the movement of the pointer. When you move the mouse across your table or desk, the pointer will move across the screen.

1. *Hold the mouse with your thumb and little finger resting on either side of the mouse. Put your index and middle fingers over the mouse buttons.*



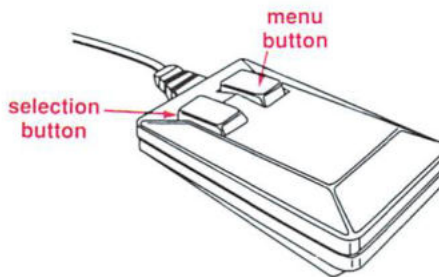
2. *Without pressing either button, slide the mouse across your desk.*

As you move the mouse, the pointer moves in the same direction. If you run out of room before getting the pointer where you want it, lift the mouse, put it down where there is more room, and continue moving it.

Lifting the mouse does not move the pointer.

To work with an icon, you must first point to it. When pointing to an icon, the tip of the pointer must be over the icon. The tip is where the pointer's "hot spot" is located.

Once you have the pointer in the right spot, you use a mouse button to send an instruction to the Amiga. There are two buttons on the mouse:



The left mouse button is the **selection button**; the right mouse button is the **menu button**. How to use the mouse buttons is explained in the following sections.

The Selection Button

The left mouse button is the selection button. You use this button to **select** icons that you want to use.

Pressing and releasing a mouse button is known as **clicking**. Clicking the selection button while pointing to an icon is known as selecting an icon. When an icon is selected, it becomes available for use and changes its appearance in some way.

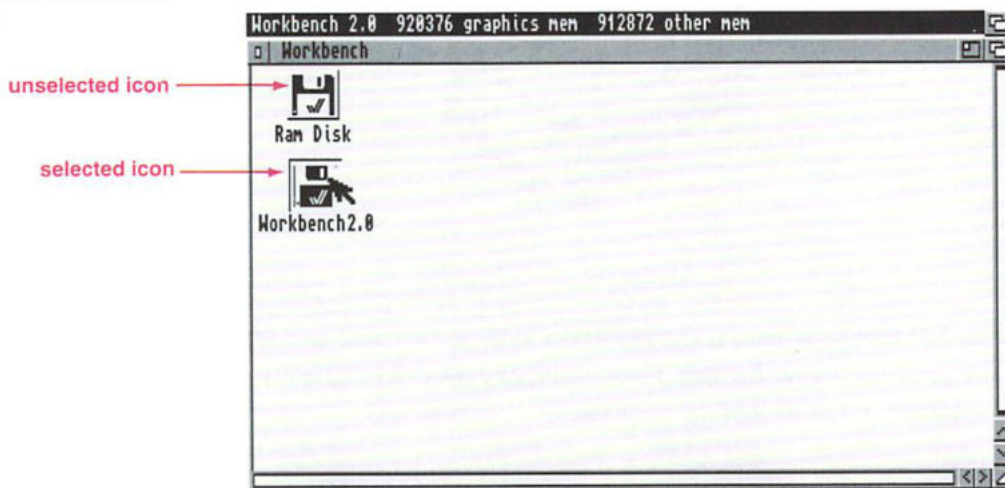
For instance, the Workbench2.0 icon, along with the other icons in the Workbench window are enclosed in a box that appears to be raised above the screen.

Throughout this chapter, you are reminded that the selection button is on the left, and the menu button is on the right. These reminders only appear in this chapter.

If you accidentally click the mouse button twice, a new window will appear on the screen. Don't worry about this. Just leave it there, and continue on with the tutorial.

1. *Point to the Workbench2.0 disk icon, and press and release the selection (left) button.*

When the Workbench2.0 icon is selected, it appears to sink into the screen. An icon will remain selected until you click on another icon or an empty part of the screen.



2. *While the Workbench2.0 disk icon is selected, point to it, hold down the selection (left) button, and move the mouse.*

A copy of the icon will move across the screen. When you release the mouse button, the icon will move to the new location. This is known as **dragging**.

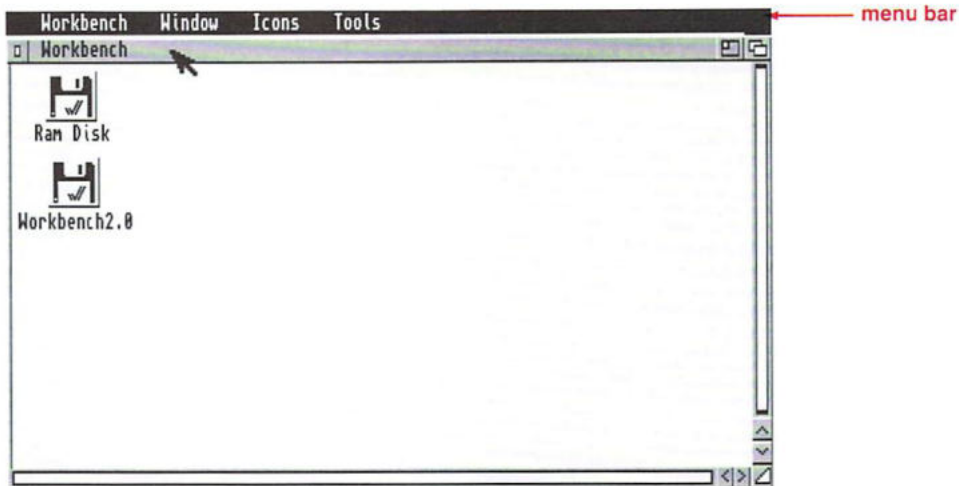
You can drag any icon by pointing to it, holding down the selection button and moving the mouse. You can also drag windows and screens. This is explained further in a later section.

The Menu Button

The right mouse button is the menu button. When this button is held down, a **menu bar** appears across the top of the screen. The menu bar shows the headings of any **menus** that are available to you. A menu is a list of options, known as **menu items**, that is provided by the software you are using.

1. Hold down the menu (right) button.

The menu bar appears across the top of the screen. It consists of four menu headings: Workbench, Window, Icons, and Tools. Each of these menus is explained in detail in Chapter 2.



The next section of this tutorial explains how to choose an item from a menu.

Quick Review

- Moving the mouse moves the pointer across the screen.
- The left mouse button is the selection button. Pressing and releasing the selection button while pointing at an icon makes that icon available for use. This is known as selecting an icon.
- Holding down the selection button while moving the mouse drags the selected icon to a new location on the screen.
- The right mouse button is the menu button. Holding down the menu button causes the menu bar to appear.

Using Menus

Most programs allow you to interact with the Amiga through menus. For instance, you can rename an icon or copy a file by choosing menu items from the various Workbench menus. While menus may vary from program to program, the steps involved in choosing an item from a menu are the same for all Amiga software.

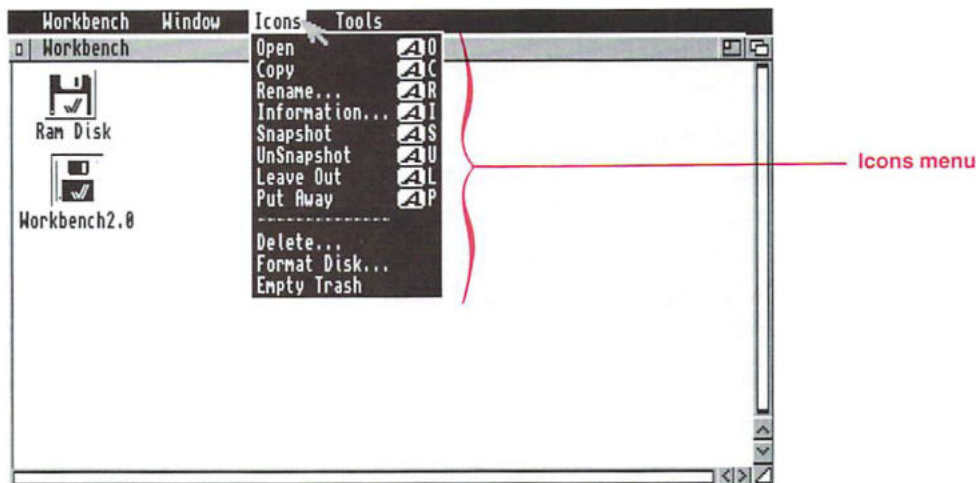
The following example shows you how to choose a menu item from the Icons menu.

1. *Point to the Workbench2.0 icon on your screen, and click the selection (left) button.*

The icon will change color and will appear to sink into the screen. This indicates that the icon is selected. (If the icon is still selected from Step 3 of the "Using the Mouse" section, it will already be highlighted. Selecting it a second time has no effect.)

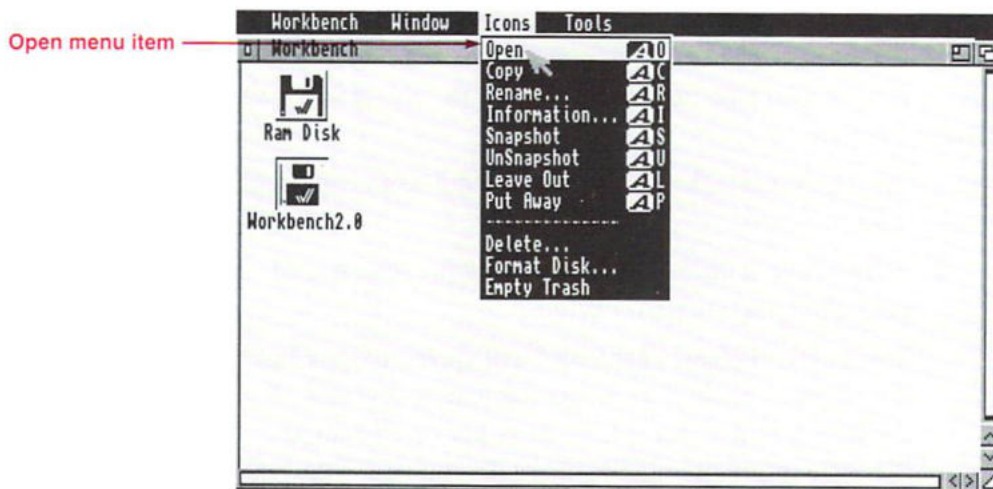
2. *Hold down the menu (right) button, and point to the Icons heading in the menu bar.*

A list of options, known as menu items, will appear.



3. *Without releasing the menu (right) button, move the pointer down to the Open menu item.*

Notice that when you point to Open, it is highlighted.



4. *While Open is highlighted, release the mouse button.*

The Workbench2.0 disk window will appear on the screen.

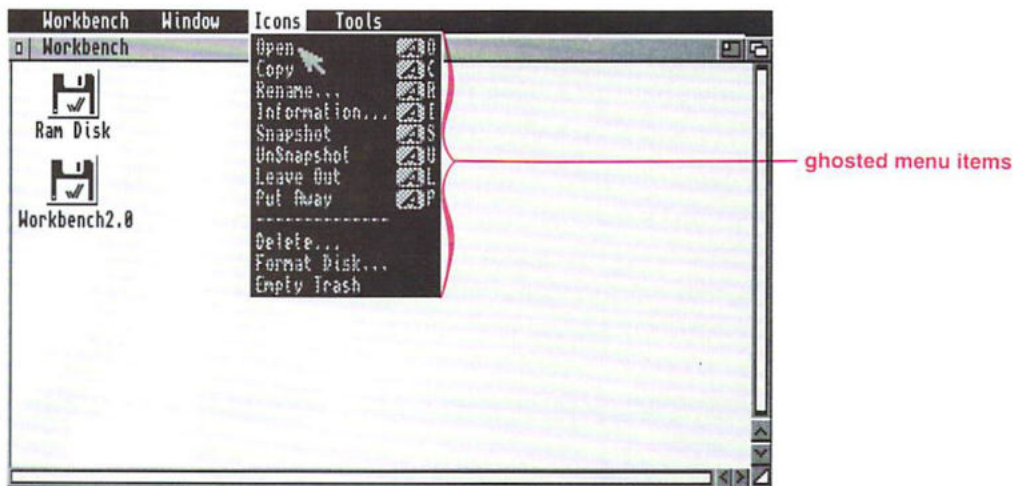
Mouse shortcut: Another way to open an icon is to point to the icon and **double-click** (quickly click twice) the selection (left) button.

Keyboard shortcut: You don't always need to use the mouse to choose a menu item. You can often get the same result by pressing two keys on the keyboard — right Amiga and a corresponding letter. Keyboard shortcuts are shown to the right of the menu item.

For instance, to choose Open from the Icons menu, you can press right Amiga-O. (Press right Amiga, keep holding it down, press O, then release both keys.)

Ghosted Menu Items

At certain times some menu items are not available for use. The unavailable items are displayed less distinctly than the others. These items are **ghosted** and will not be highlighted when the pointer passes over them.



Usually menu items are ghosted because something on the screen must be selected before the menu can be used. For instance, if you haven't selected an icon, you cannot choose any of the menu items in the Icons menu.

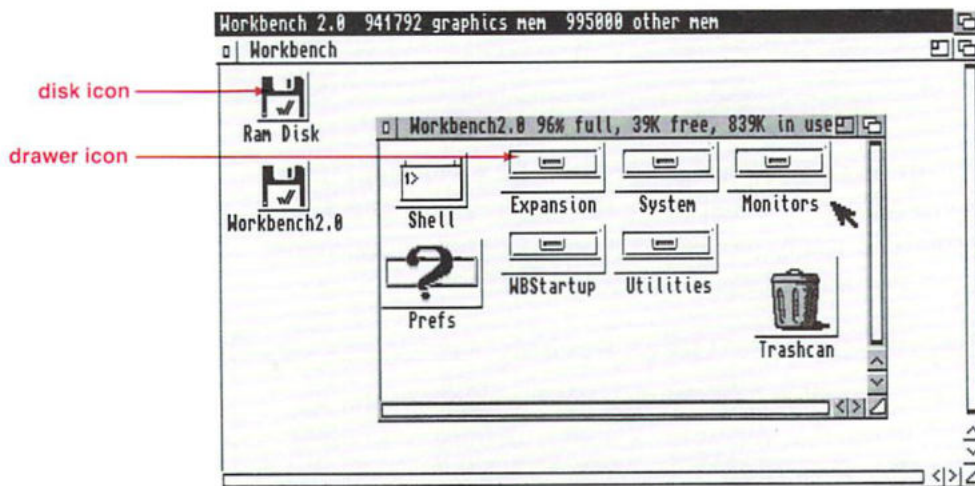
Quick Review

- Before choosing a menu item, the icon, window, or screen that you want to work with must be selected.
- To choose the menu item, hold down the menu (right) button, highlight the menu item by pointing to it, and release the menu button.

Working with Windows

When you open a disk or drawer icon, a window appears. A window is a rectangular area on the screen that displays information, such as graphic images and text, and allows you to work with that information. If you were following the steps explaining how to use the menus, you should have two windows on your screen: the Workbench window and the Workbench2.0 disk window.

The Workbench window contains icons for the Ram Disk, for floppy disks that have been inserted into a disk drive, and for any hard disks that are installed in the system. (You can also add other types of icons to the window. This is explained in the "Icons Menu" section of Chapter 2.)



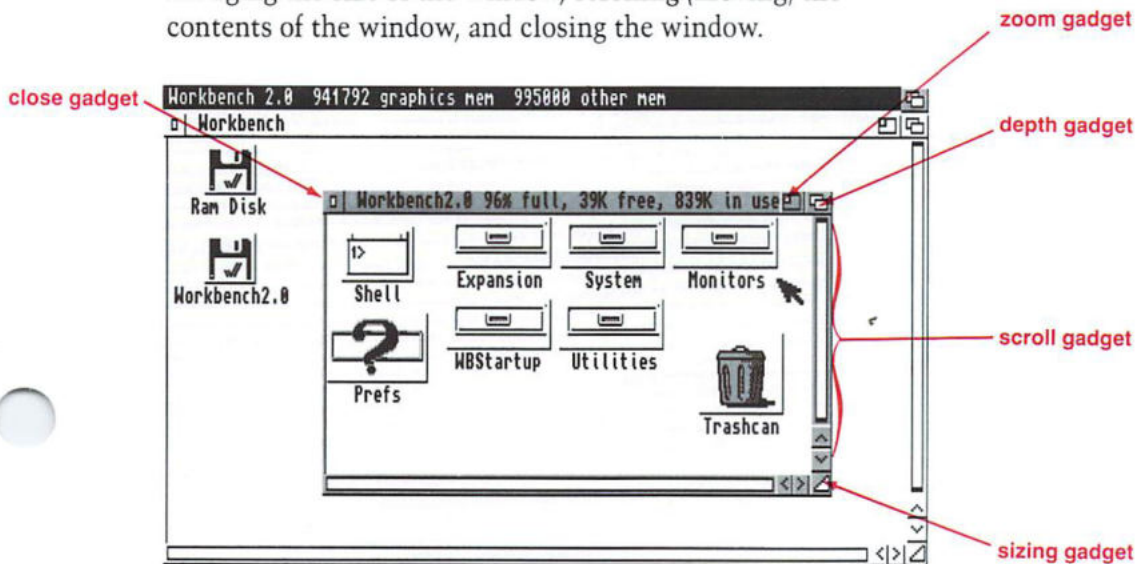
The Workbench2.0 disk window contains several **drawer** icons. Drawers are subdivisions of a disk that are used to organize the information on that disk.

If you have a hard disk system, the System2.0 disk window contains drawers for both the Workbench2.0 and Extras2.0 disks. While your window contents may look slightly different than the windows shown in this manual, you can still follow this tutorial.

The Amiga is a **multitasking** computer. This means that it is possible to have more than one program running at a time. For instance, you could have a calculator, a word processing program, and a spreadsheet all open and running at the same time. This usually results in several windows open at once, with the windows overlapping each other.

However, only one window at a time can be **active** (capable of accepting information). The frame of the active window is a different color from the other windows on the screen.

Take a look at the border of the Workbench2.0 disk window. It contains several boxes, known as **gadgets**, that let you control the window in many ways, including moving the window, changing the size of the window, **scrolling** (moving) the contents of the window, and closing the window.



These are some of the most common gadgets found in windows on the Amiga. Depending on the software you are using, windows can contain any combination of these gadgets. Some windows have gadgets particular to their software. For instance, a window in a paint program may have gadgets that let you choose colors, brushes, magnifications, or other specialized functions.

The gadgets in the Workbench window are briefly explained in the following sections. Complete information on standard system gadgets is available in Chapter 2.

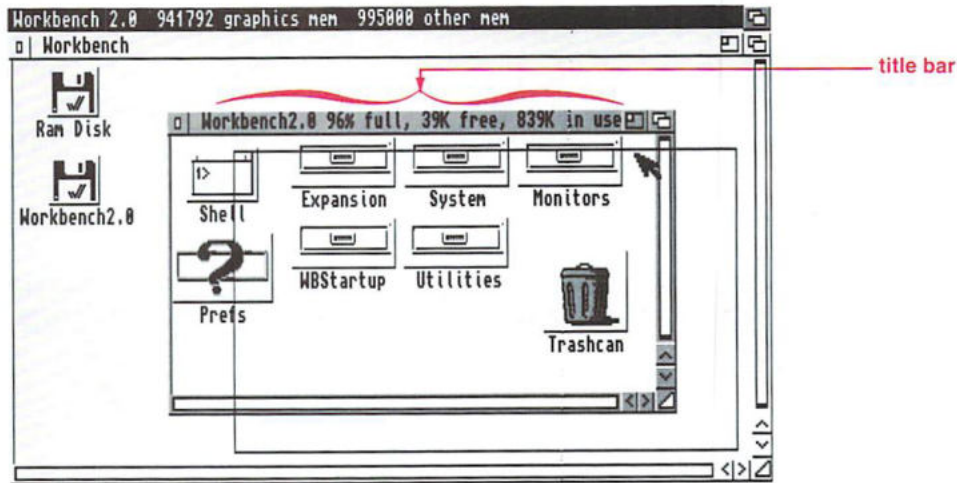
The Title Bar

The **title bar** identifies each window. The title bar of the Workbench2.0 disk window contains the name of the window and information as to the amount of data contained on the Workbench2.0 disk. When a drawer icon is opened, the title bar of the drawer window displays the name of the drawer.

You can also use the title bar to drag a window across the screen.

1. *Point to the title bar of the Workbench2.0 disk window.*
2. *Hold down the selection (left) button, and move the mouse.*

An outline of the window will appear and will move in the same direction as the mouse.



3. Release the mouse button.

The window will move to its new location.

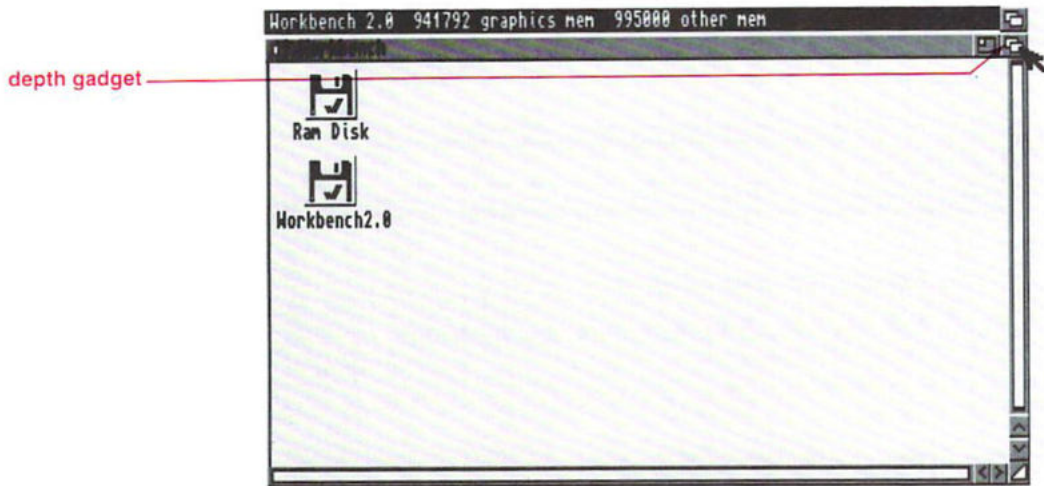
The Depth Gadget

The **depth gadget** lets you move a window to the front of the screen or push it behind any other open windows. This is useful when you have several windows open and the one you need is not at the front of the screen.

If you've been following this tutorial, there should be two windows open on your screen: the Workbench window and the Workbench2.0 disk window.

1. *Point to the depth gadget on the Workbench window, and click the selection (left) button.*

The Workbench window moves to the front of the screen. The Workbench2.0 disk window has not been closed. It is just hidden behind the larger Workbench window.



2. *Point to the depth gadget on the Workbench window, and click the selection (left) button again.*

The Workbench window will be pushed to the back of the display, and you will be able to see the Workbench2.0 disk window again.

The action of the depth gadget depends upon the window's location. If the window is behind another window, selecting the depth gadget brings it to the front of the screen. If the window is at the front of the screen, selecting the depth gadget sends it to the back. You can also send a window to the back of the screen, regardless of its current position, by holding down Shift and selecting the depth gadget.

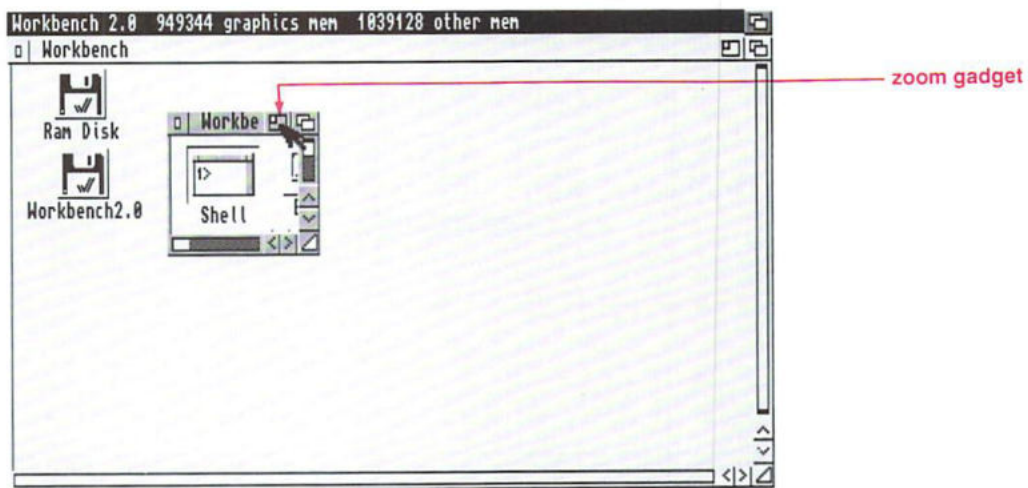
The Zoom Gadget

The **zoom gadget** changes the size of a window. This is helpful when you have several windows open on the screen. You can move unneeded windows out of the way by making them smaller with the zoom gadget.

Selecting the zoom gadget on the Workbench2.0 disk window reduces the size of the window.

1. *Point to the zoom gadget in the Workbench2.0 disk window, and click the selection (left) mouse button.*

The Workbench2.0 disk window becomes smaller.



2. *Click on the zoom gadget again.*

The Workbench window returns to its previous size and position.

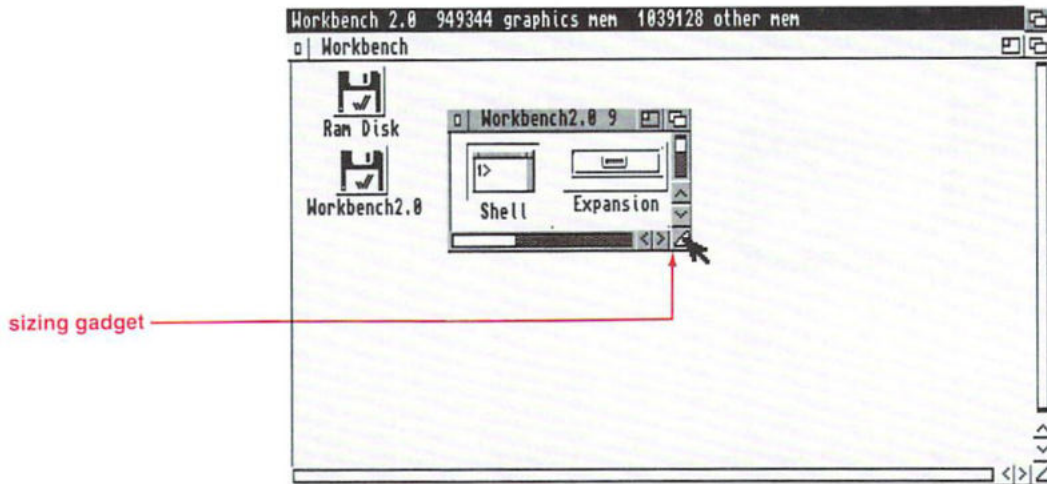
In general, if a window is small when it is opened, selecting the zoom gadget will make it large. If a window is large when it is opened, selecting the zoom gadget usually makes it smaller. If you change the size or position of a window, the window will use that new size or position when the zoom gadget is selected.

The Sizing Gadget

Another way to change the size of a window is by using the **sizing gadget**. This is an easy way to change a window's size so that you can see other information on the screen.

1. *Point to the sizing gadget in the lower right corner of the Workbench window.*
2. *Hold down the selection (left) button, and move the pointer up and to the left.*

The window becomes smaller as you move the mouse.



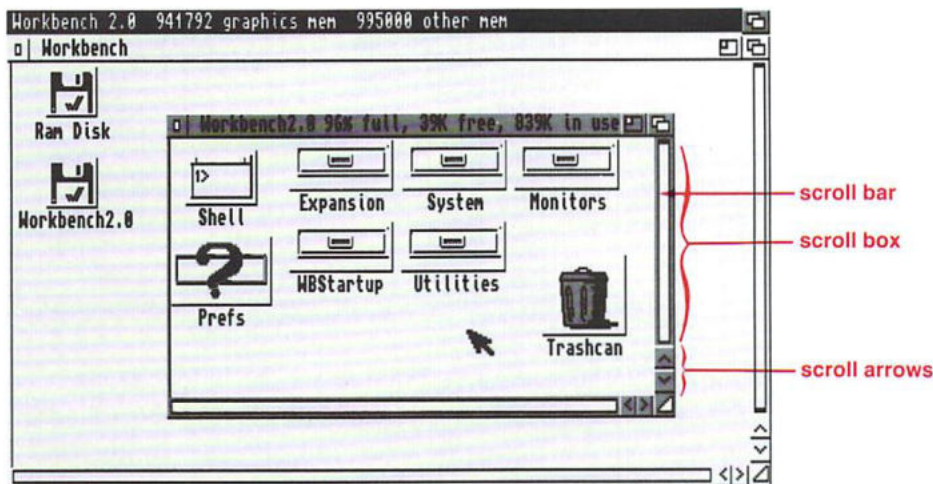
3. *Release the mouse button.*

The window stays the smaller size.

You can enlarge the window by pointing to the sizing gadget and dragging the window down and/or to the right.

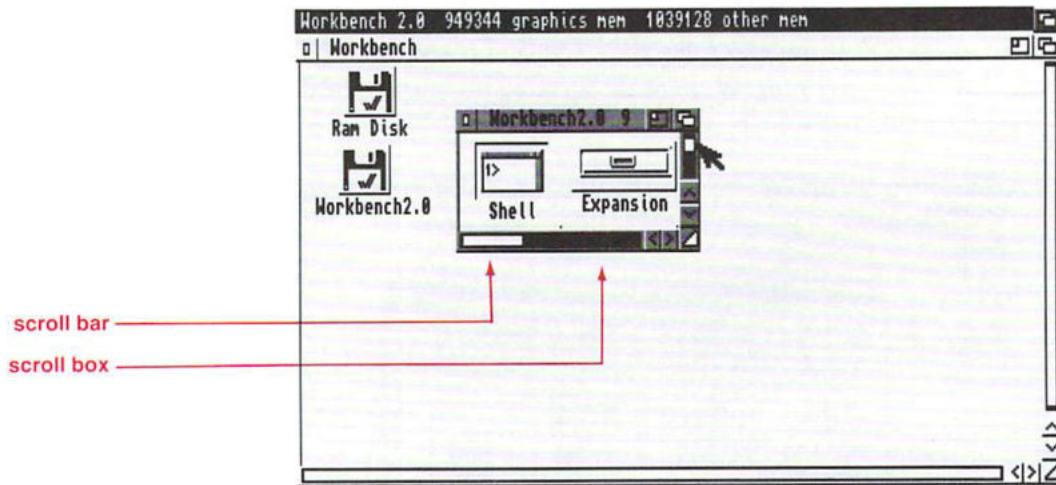
The Scroll Gadgets

When a window is small, you cannot always see all its icons. You can tell whether all the icons are visible by looking at the **scroll gadgets** that run along the right side and bottom of the window. The scroll gadgets are made up of scroll boxes, scroll bars and scroll arrows.



The **scroll bars** are the highlighted areas inside the **scroll boxes**. The bars change size depending on how much of the window's contents are visible. When the scroll bar completely fills a scroll box, all of the icons are visible.

When some of the icons in the window are not visible, the scroll bars shrink and only fill part of the scroll box.

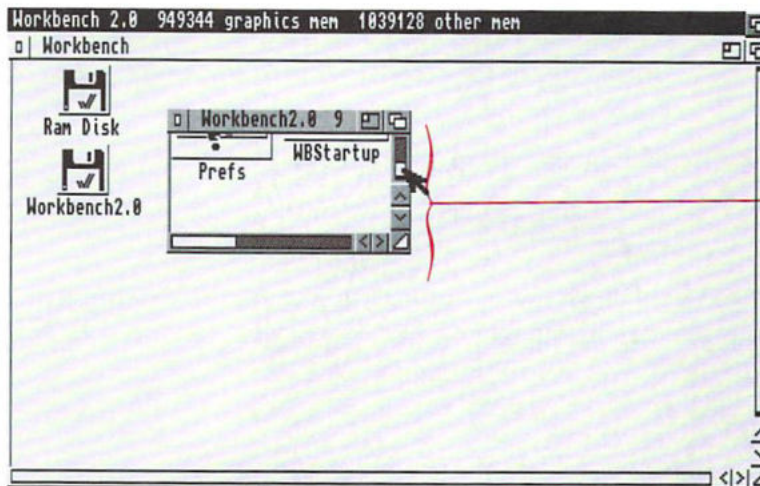


By dragging the scroll bars, you can see the hidden icons without changing the size of the window. If you were following the steps on how to use the sizing gadget, your Workbench2.0 disk window should be small, as shown above. If not, use the sizing gadget to make the window smaller.

1. *Point to the scroll bar along the right side of the window.*

2. *Hold down the selection (left) button, drag the scroll bar down, and release the mouse button.*

Drag the scroll bar into the empty space of the scroll box. For instance, if the bottom of the scroll box is empty, drag the scroll bar down so that the bottom of the box is full and the top is empty. When you release the mouse button, you will see the icons from the bottom of the window.



bottom left
corner of
window is
visible

3. *Point to the scroll bar in the bottom of the window.*
4. *Hold down the selection (left) button, and drag the scroll bar to the right.*

When you release the mouse button, you can see the icons from the right side of the window.

The location of the scroll bar indicates what part of the window is currently visible. For instance, if the scroll bar is in the lower portion of the right scroll box, you are looking at the bottom of the window.

Another way to move the scroll bar is to point in the empty area of the scroll box and click the selection (left) button. The scroll bar will move to the empty area.

The **scroll arrows** also allow you to scroll through the viewing area of the window. The direction of the arrow determines in which direction the viewing area will move.

1. *Point to one of the scroll arrows in the lower right corner of the Workbench2.0 disk window.*
2. *Click the selection (left) button.*

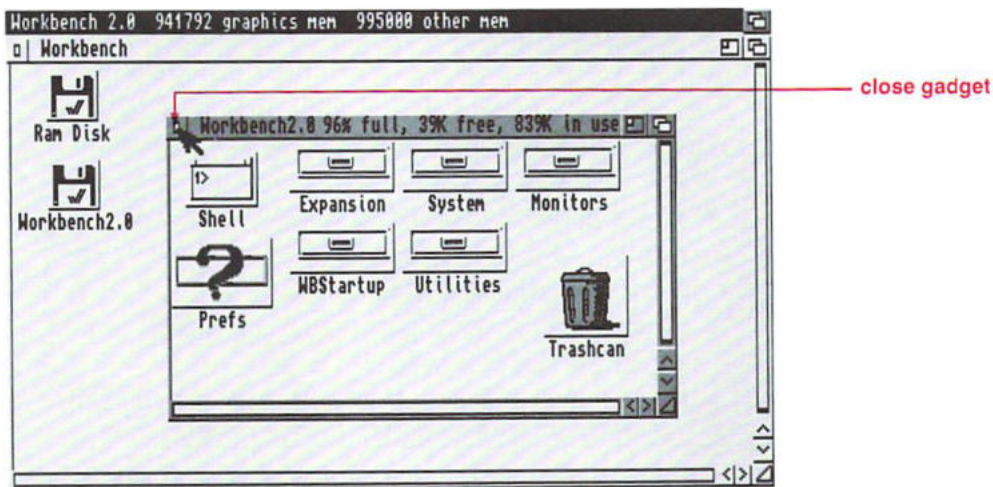
The viewing area will move in the direction of the arrow.

If you hold down the selection (left) button, the viewing area will move more quickly.

The Close Gadget

When you are finished working in a window, you can use the **close gadget** to remove it from the screen.

1. *Point to the close gadget in the Workbench2.0 disk window.*



2. *Click the selection (left) button.*

The Workbench2.0 disk window disappears.

Be especially careful of the close gadget on the Workbench window. When you select this gadget, a **requester** asks you if you are sure you want to quit the Workbench. If you close that window, you cannot access any of the Workbench programs.

Note: You do not want to close the Workbench at this time. Cases in which you may want to do so are explained later in this manual.

A requester is a message from the system. It is a box that appears on the screen and has gadgets that let you select a course of action.

Quick Review

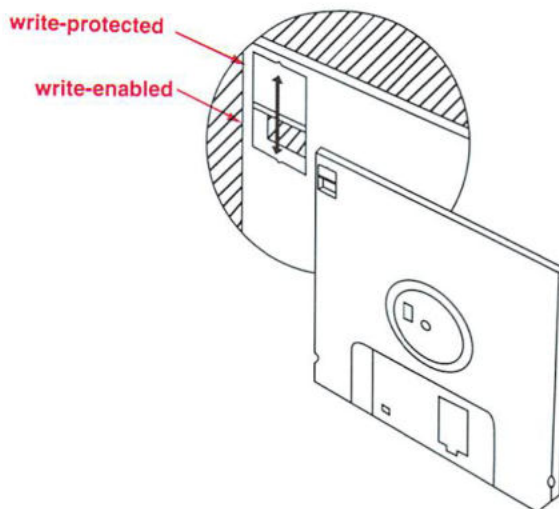
- The title bar displays the name of the window and information about its contents. By pointing to the title bar, holding down the selection (left) button and moving the mouse, you can drag a window to another area of the screen.
- Selecting the depth gadget of the front-most window moves that window behind any other open windows. If a window is behind another window, selecting its depth gadget brings it to the front of the screen. Holding down Shift and selecting the depth gadget sends a window to the back of the screen.
- Selecting the zoom gadget changes the size of a window. Selecting it again returns the window to its previous size and position.
- By dragging the sizing gadget, you can expand or shrink the size of a window.
- Dragging a scroll bar enables you to see the hidden areas of a window. Another way to move a scroll bar is by pointing to the empty area of the scroll box and clicking the selection (left) button.
- Selecting a scroll arrow scrolls the viewing area of the window in the direction of the arrow. Pointing to a scroll arrow and holding down the selection (left) button moves the viewing area more quickly.
- Selecting the close gadget removes a window from the screen.

Making Backup Copies of Disks

One of the most important things for you to do is to make **backup** copies of all your disks. A backup copy is simply a duplicate of an original disk. It is important to use the backup as your everyday working disk and to store the original disk in a safe place. This way if the working disk is ever damaged, you can make another copy from the original disk.

Most application software allows you to make a backup copy. Generally when you purchase a program, it includes a licensing agreement. Be sure to read the agreement to learn exactly how many copies you are allowed to make. Making and distributing unlicensed copies of disks is a copyright violation (also known as software piracy) and is illegal.

To copy your Workbench2.0 and Extras2.0 disks, you need two blank 3.5 inch disks that are **write-enabled** (able to accept information). This means that the small plastic tab in the corner of the disk must be covering the hole.



When copying your disks, you'll notice that the Amiga often instructs you to insert disks into specific disk drives. It refers to these drives by their **drive names**. The drive names for the various Amiga models are shown below.

Amiga Drive Names		
Location of Drive	A500	A2000/A3000
1st internal drive	DF0:	DF0:
2nd internal drive	n/a	DF1:
1st external drive	DF1:	DF2:
2nd external drive	DF2:	DF3:

You can refer to a disk by its **volume name**, the name that is under its icon, or you can refer to a specific disk drive by its drive name. For instance, if you tell your Amiga to copy the disk in DF1:, it will copy whatever disk is in drive DF1:. (A colon must always follow the volume or drive name.) Be careful when specifying a drive name when saving or copying files. Make sure you know which disk is in which drive or you could accidentally save or delete files on the wrong disk.

When copying your disks, you will review most of the basics that you have just learned. If you only have one floppy drive, please read the next section. If a second floppy drive has been added to your Amiga, skip ahead to the section titled "Using Two Disk Drives," on page 1-33.



Even if you have a hard disk system, you should still make backup copies of your original disks just to be safe. This will also give you practice in copying disks.

Using One Disk Drive

When you use only one disk drive to copy your disk, the system reads information from the Workbench2.0 disk (the **source** disk) into the Amiga's internal memory. Then, you have to remove the Workbench2.0 disk from the disk drive and insert a blank disk (the **destination** disk). The information is then written to the blank disk. This is known as a disk **swap**. You may have to swap disks several times before the copy is completed. The steps involved are outlined below.

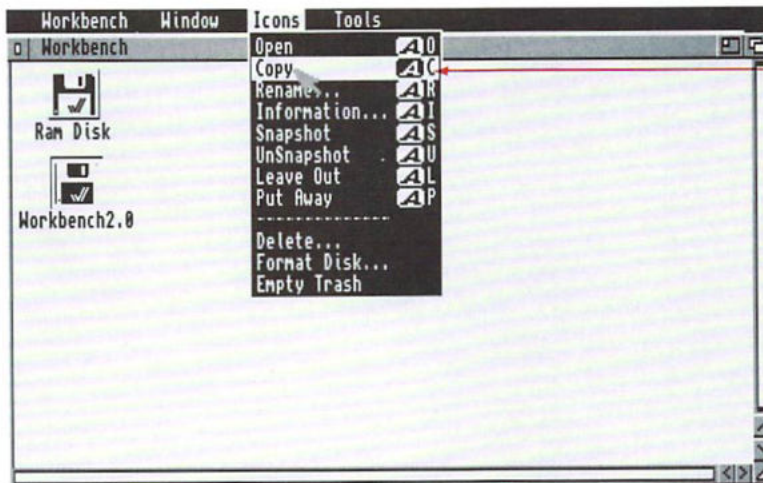
1. *The Workbench2.0 disk should be in the internal disk drive, known as DF0:.*
2. *Point to the Workbench2.0 icon, and click the selection (left) button.*

The icon is now highlighted.

3. *Hold down the menu (right) button.*

The menu bar appears.

4. *Point to the Icons menu, move the pointer down to the Copy menu item, and release the menu button.*



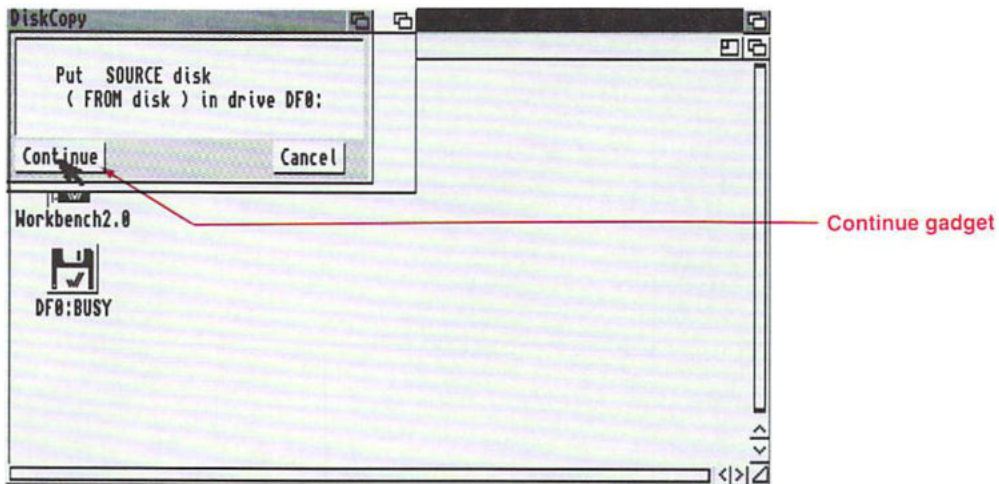
Copy menu item

A requester appears and shows how many times you will have to swap the disks. The number of swaps depends on the amount of memory available to the Amiga. If you have enough memory in your system that the disk copy will take fewer than 5 swaps, this requester may not appear.



5. *Point to the Continue gadget, and click the selection (left) button.*

A second requester asks you to insert the source disk, Workbench2.0, into drive DF0:, the internal disk drive.



6. Since the disk is already in the drive, simply point to the Continue gadget in the requester, and click the selection (left) button.

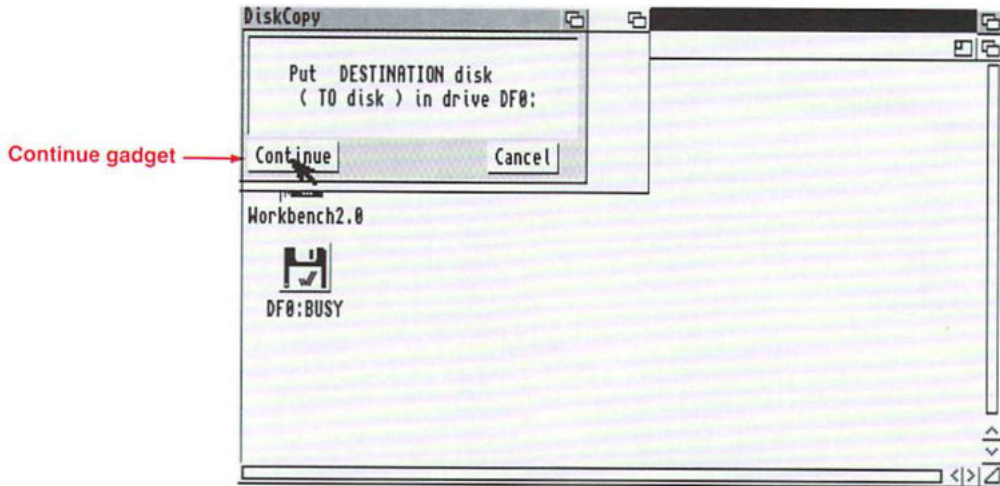
The text in the requester will show how many cylinders have been read and how many are left.

After the Amiga has read as much information as it can from the source disk, a third requester instructs you to insert the destination disk into drive DF0:.

Make sure the disk drive light is out before removing the Workbench2.0 disk from the drive.

A cylinder is a physical division of a disk. The 3.5 inch disks used with the Amiga have 80 cylinders numbered 0-79.





7. Put your blank disk into the drive, then select the Continue gadget.

Point to the Continue gadget in the requester, and click the selection (left) button. The data that was read into memory is copied to the blank disk.

To finish copying the disk, follow the requesters that appear and switch back and forth between the Workbench2.0 disk and the backup disk as many times as requested by the system. (Be sure the drive light is out before ejecting a disk from the disk drive.) When the copy is finally finished, the message Disk Copy Finished appears on the screen.

8. Remove the backup disk from the drive and put an adhesive label on it.

Write the name of the disk on the label.

The procedure is the same for copying the Extras2.0 disk except for one small detail. After you choose Copy from the Icons menu, a requester asks you to insert the Workbench2.0 disk into any drive. The Amiga must load the DISKCOPY program on the Workbench2.0 disk before it can begin copying the Extras2.0 disk.

At this point, be sure to use your original Workbench2.0 disk. A second requester asks for the Extras2.0 disk, and from there the procedure follows the steps outlined above. Always be sure to read the exact text in a requester and follow the instructions.

Hard Disk users will not see the requester that asks for Workbench2.0, since the Amiga can read the necessary program from the hard disk.

The backup disks are named copy_of_Workbench2.0 and copy_of_Extras2.0. You'll learn how to rename the disks on page 1-38.



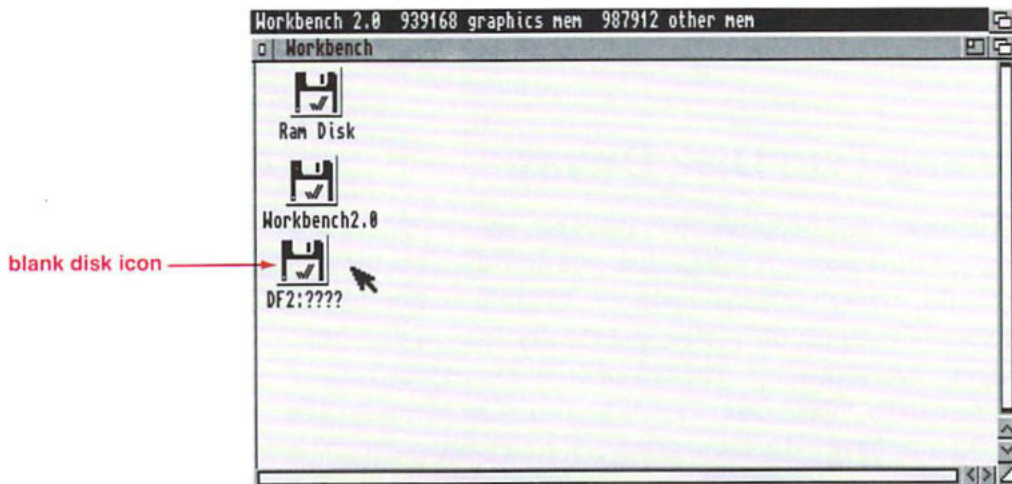
Using Two Disk Drives

When only one floppy drive is available, the system reads information from the **source** disk (the disk that is being copied) into the Amiga's internal memory. Then the source disk is removed from the disk drive, and the **destination** disk (the blank disk) is inserted. The information in memory is then copied to the blank disk. This is known as a disk **swap**. The disks may have to be swapped several times, depending on how much memory is available.

When you have a second disk drive added to your Amiga, you can save time by putting the Workbench2.0 disk (the source disk) into one drive and the blank disk (the destination disk) into the other. This way the Amiga can copy directly from one disk to the other. This eliminates the need to swap disks and allows the disk duplication to proceed much faster. The steps involved are outlined below.

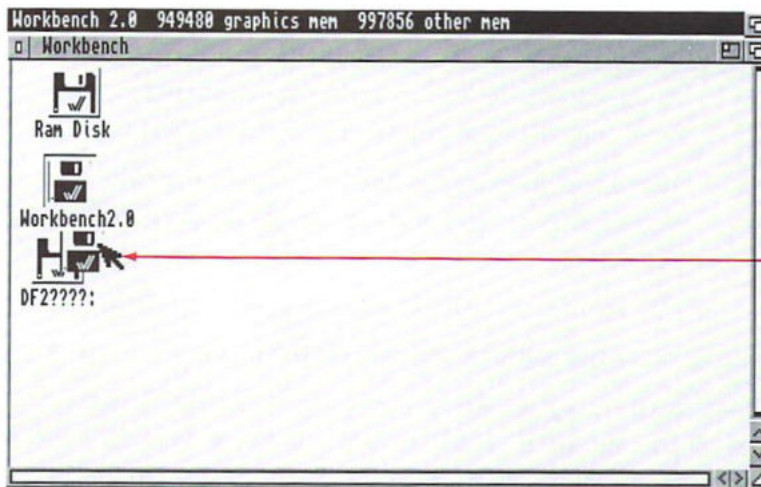
1. *Insert the Workbench2.0 disk into the Amiga's original internal drive, known as DF0:.*
2. *Insert the backup disk into your second drive.*

A new disk icon appears on the screen. Since it is a brand-new disk, the icon is labeled DF1:???? or DF2:????, depending on which disk drive it is in. At this point, there is no information on the disk. Therefore, the computer sees it as an unknown disk.



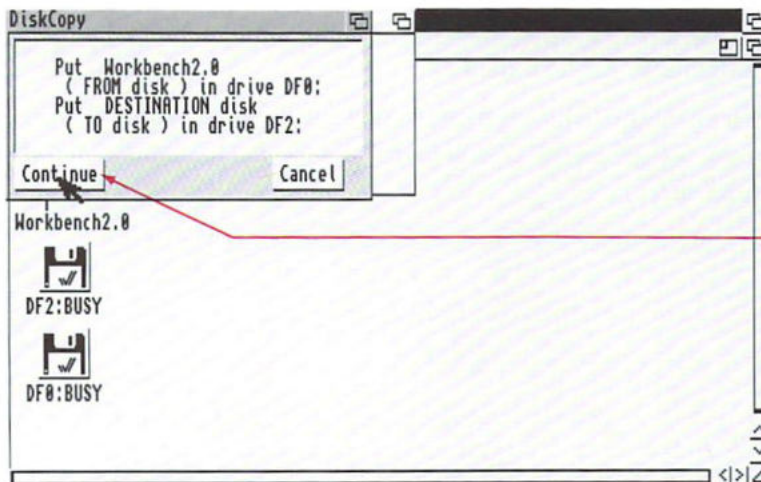
3. *Point to the Workbench2.0 disk icon, and press the selection (left) button.*

4. Continue to hold down the selection (left) button, and drag the Workbench2.0 disk icon over the icon for the blank disk.



5. Release the mouse button.

A requester will ask you to insert Workbench2.0 into drive DF0: and to insert the destination disk into your second drive (DF1: or DF2:, depending on your system configuration).



A cylinder is a physical division of a disk. The 3.5 inch disks used with the Amiga have 80 cylinders, numbered 0 to 79.

6. *Since the disks are already inserted, simply point to the Continue gadget in the requester and click the selection (left) button.*

The Amiga will read the information on the Workbench2.0 disk and copy it to the destination disk. A requester shows the number of **cylinders** that have been copied and the number of cylinders left to be read.

7. *When the disk copy is finished, make sure that both drive lights are out and remove the backup disk from the drive.*
8. *Put an adhesive label on the new backup disk.*

Write the name of the disk on the label.

The procedure is the same for copying the Extras2.0 disk except for one small detail. After you drag the Extras2.0 disk icon over the blank disk's icon, a requester asks you to insert the Workbench2.0 disk into any drive. The Amiga needs to load the DISKCOPY program from the Workbench2.0 disk before it can begin copying the Extras2.0 disk. A second requester asks you to replace the Extras2.0 disk, and from there the procedure follows the steps outlined above.

Hard Disk users will not see the requester that asks for Workbench2.0, since the Amiga can read the necessary program from the hard disk.

The backup disks are named copy_of_Workbench2.0 and copy_of_Extras2.0. You'll learn how to rename the disks in the next section.



Quick Review

With one disk drive:

Select the icon for the disk to be copied. Choose Copy from the Icons menu. You'll have to swap back and forth between the source disk (the disk being copied) and the destination disk. Follow the instructions in the requesters that appear on the screen. When the Disk Copy Finished message appears, wait for the disk drive light to go out, then remove the disk from the drive.

With two disk drives:

Drag the icon for the disk to be copied over the icon of the destination disk. A requester will appear confirming that the disks are in the appropriate drives. Select Continue, and the disks will be copied. Be sure all disk drive lights are out before removing the disks from the drives.

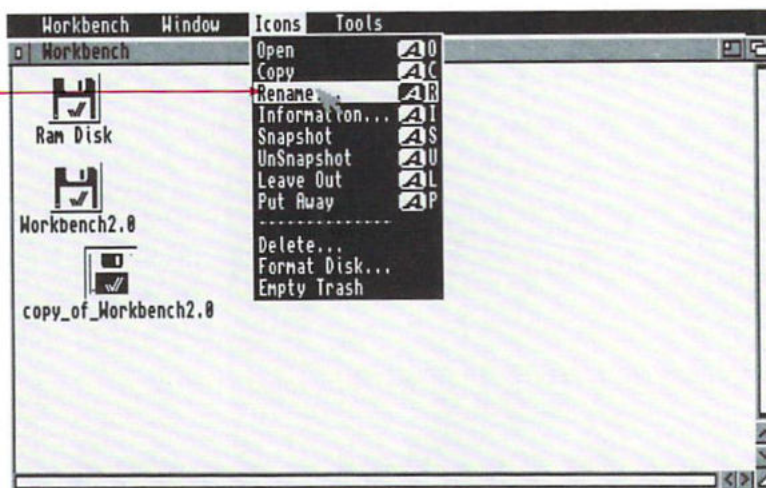
Renaming Your Backup Disks

After copying your Workbench2.0 and Extras2.0 disks, you should rename the backup disks, removing the words `copy_of_` from the names. To rename a disk:

1. Put the `copy_of_Workbench2.0` disk in the disk drive.
2. Point to the `copy_of_Workbench2.0` icon, and click the selection (left) button.
3. Hold down the menu (right) button, point to the Icons menu heading, then move the pointer down to the *Rename* menu item.

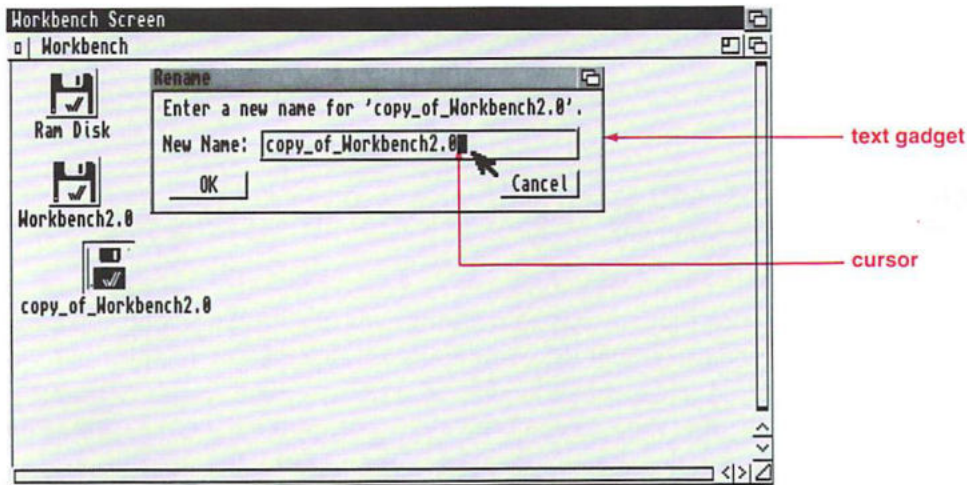
Rename is highlighted.

Rename menu item



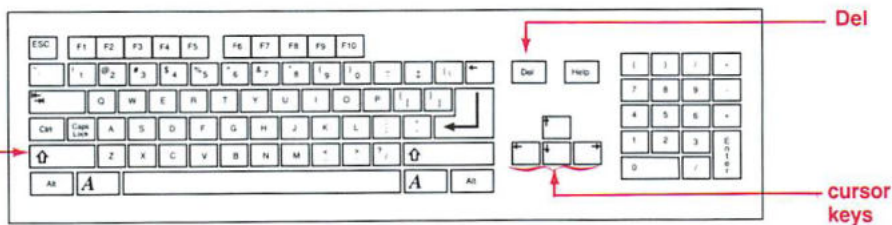
4. Release the menu (right) button.

A requester will appear containing the words
copy_of_Workbench2.0.



5. Delete copy_of_.

Move the cursor to the beginning of the text gadget. To do this you can use the left cursor key, press Shift-left cursor, or click on the C in copy. Then use Del to erase copy_of_. Each time you press Del, the character under the cursor will be deleted.



Be sure to erase the underscore before the W.

6. Press Return.

The requester disappears and the disk icon is now named Workbench2.0.

Follow the same steps to rename copy_of_Extras2.0 to Extras2.0.

Rebooting the Amiga

Now that you have a working copy of your Workbench2.0 disk, you can **reboot** the computer with that disk. Reboot is the term used to describe the process of resetting the Amiga without turning the power off. Rebooting eliminates any data stored on the Ram Disk and abruptly terminates any programs that are running.



Hard disk users can reboot directly from the hard disk. If the Workbench2.0 disk is in your floppy drive, remove it. Otherwise, the Amiga will boot from the floppy disk instead of the hard disk. Skip to step 2.

1. Insert your working copy of the Workbench2.0 disk into the Amiga's disk drive.

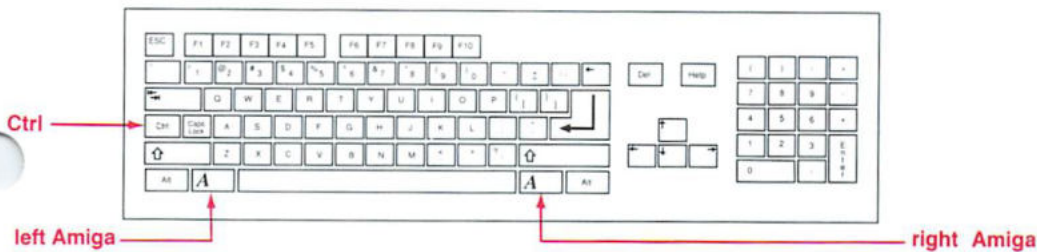
If your Amiga has more than one floppy disk drive, it does not matter which drive you use. The Amiga checks each drive to see which one contains a **bootable** disk. (A bootable disk is one that contains the files the Amiga needs to start operation.) However, if the Amiga does

not find a bootable disk in any drive, it displays the animated screen.

At this point, a bootable disk must be inserted into drive DF0: in order for the Amiga to boot. The Amiga will not check any of the other drives after the initial search.

2. *Wait for the disk drive light to go out, then hold down Ctrl (Control), left Amiga, and right Amiga.*

All three keys must be held down at the same time.



Just as you should never remove a disk from the drive while the floppy drive light is on, you should also never reboot or turn off the Amiga when a floppy disk or hard disk drive light is on. Be sure to wait for all disk activity to stop.



The Workbench screen will appear just as it did when you booted with the original Workbench2.0 disk. Once you have rebooted with your working copy of Workbench, store the original disk in a safe place.

Using Application Software

Application software refers to the programs available for use with your Amiga, such as databases, video and sound programs, or CAD (computer-aided design) packages. When you purchase application software, be sure to read the documentation included with the software to learn how to use the program.

Some application software is supplied on a bootable disk. This means that you can insert the program disk into the Amiga's disk drive, turn on (or reboot) the Amiga, and get started. You usually do not need to use the Workbench2.0 disk.

Just as you made a backup of your Workbench2.0 disk, you should always make a backup of your program disks. (Most application software will allow you to do this, although some programs may be copy-protected. Some programs may also place restrictions on your right to make backups. Please consult your license agreement packaged with the application.) Store the original program disk in a safe place, and use the copy as your working disk. This way if anything ever happens to damage the disk, you can make another copy from the original.



Hard disk users should consult the program's documentation (and Chapter 6 of this manual) to learn how to copy the program to their hard disk. This allows you to run the program directly from the hard disk instead of from the floppy disk, increasing the speed of operation.

When you boot with a program disk, the program may automatically start (this is common when loading games, animated demonstrations, etc.), or you may be presented with a Workbench screen. If a Workbench screen appears, you may need to open the program disk icon, then another program icon to get started. Again, this procedure varies, so be sure to read the documentation packaged with the program.

No matter what type of application software you are using, you will undoubtedly want to save your work. In general, it is a good idea to save your work on disks other than the program disks. You should also make a copy of your original program disk(s), and use the copy as your working disk.

For instance, when using a word processing program, you will want to keep your **data** files on a separate disk, or data disk. Your data files contain the documents you create with your word processor. One reason for using a separate disk is that there may not be much extra room on the program disk.

Another reason is to safeguard your program disk against accidental damage. If possible, you should keep the disk **write-protected**. The small plastic tab in the corner of the disk should not be covering the hole. This way you can't erase or write over any files stored on the disk.

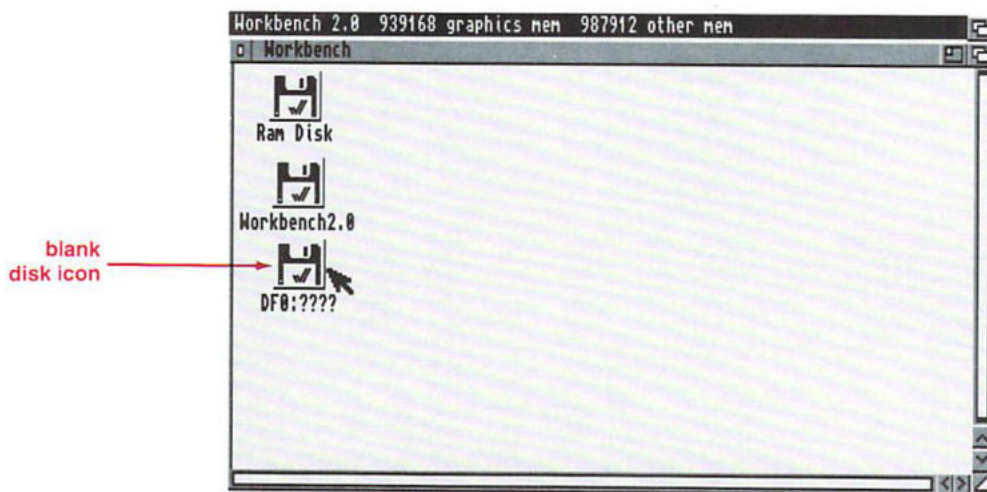
Formatting a Disk

Before you can use a disk for data storage, you must first **format** the disk. The magnetic surface of a blank disk is one large continuous area. When a disk is formatted, the Amiga's operating system (AmigaDOS) divides that area into manageable sections so that it can easily find stored information. An easy way to do this is with the Format Disk menu item in the Icons menu.

The following steps explain how to format a blank disk. You may want to try this now so that you know how to do it in the future. Or, you may want to come back to this section when you need a formatted disk. It might be a good idea to format at least one disk now, so that you have one readily available when you need it.

1. *Insert a blank disk into the disk drive.*

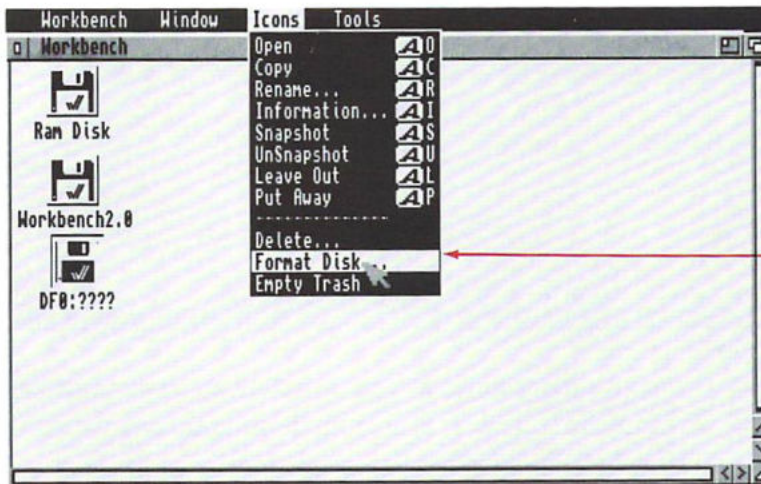
The disk icon is labeled DF0:???, DF1:???, or DF2:???, depending on which disk drive it is in. Make sure the disk is write-enabled.



2. Point to the disk icon, and click the selection (left) button.

The icon will be highlighted.

3. Hold down the menu (right) button, point to the Icons menu, move the pointer down to Format Disk, and release the menu (right) button.



Format Disk
menu item

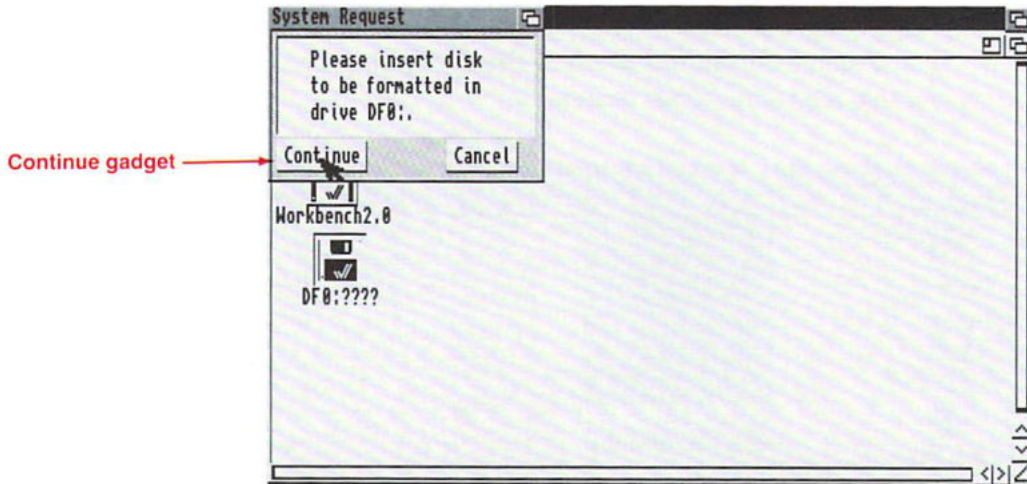
4. A requester will ask you to insert the Workbench2.0 disk in any drive. Insert the Workbench2.0 disk, point to the Continue gadget, and click the selection (left) button.

The Amiga must load a program from the Workbench2.0 disk before it can format the blank disk.

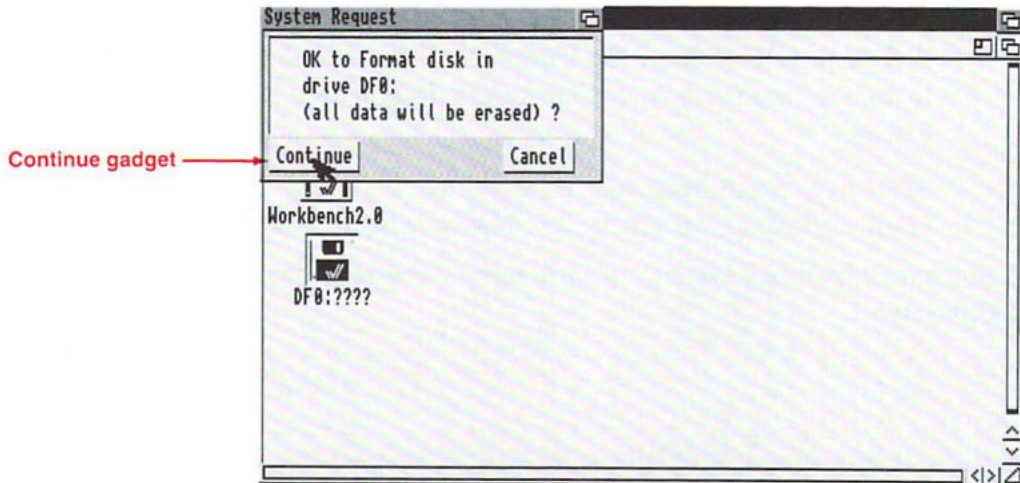
If you have a hard disk system, you will not see this requester. Skip to step 5.



5. A requester asks you to insert the disk to be formatted into the disk drive. Insert the disk, point to the Continue gadget and click the selection (left) button.



Next, a requester asks you if it is OK to format the disk in the disk drive. It also reminds you that any data on the disk will be erased.

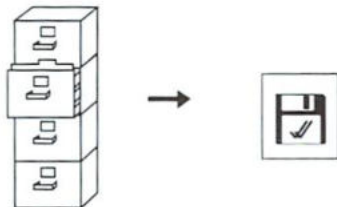


6. *Point to the Continue gadget in the requester, and click the selection (left) button.*

Once the formatting process begins, you'll see text in the requester showing the cylinder of the disk that is being formatted and verified. When the disk is formatted, its disk icon is labeled Empty. You can change this to any name you like by using the Rename item in the Icons menu.

Organizing Information on a Disk

To store information on a disk in a logical manner, disks are generally divided into drawers. Think of a disk as a filing cabinet. You wouldn't take all your papers and throw them into the cabinet. You would put drawers in the cabinet. Then you would organize the papers into folders and put the folders in the drawers.



A formatted blank disk does not contain any drawers. If you were to store all of your files on the formatted disk without creating any drawers, it would be like throwing your papers into the drawerless file cabinet. When you opened the disk window, all the file icons would be in the disk window. If you had many icons on the disk, you would have to scroll through the window until you found the correct icon. This could be quite tedious and time consuming. Instead you should create some drawers, then put your files into them.

Let's assume that you have several business reports you want to keep on disk. Perhaps for each month you produce several reports pertaining to payroll, inventory, sales, office expenditures, etc. Here's how you might organize them:

First, you should give your disk an appropriate name. For this example, we'll name it Reports. To name the disk, use the Rename item in the Icons menu.

1. Insert the formatted disk into the disk drive.

The formatted disk is labeled Empty.

2. Select the Empty disk icon.

Point to the icon, then click the selection (left) button.

3. Hold down the menu (right) button, point to the Icons menu heading, then move the pointer down to the Rename menu item.

Rename is highlighted.

4. Release the menu button.

A requester containing the word Empty appears on the screen. Use Backspace to erase the word Empty. Type the name Reports.

5. Press Return.

The disk icon is now labeled Reports.

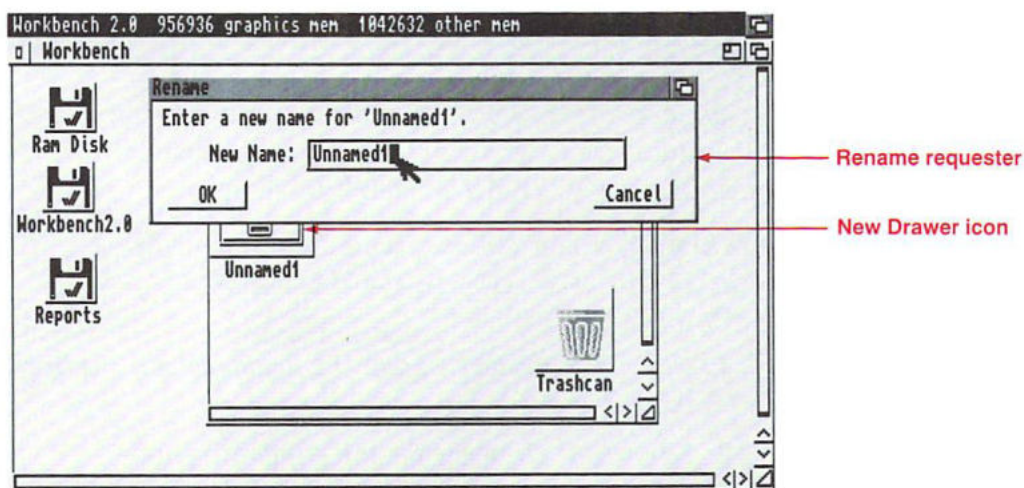
Next, you might create four drawers pertaining to the four quarters of the year: Quarter1, Quarter2, Quarter3, and Quarter4. You can create drawers with the New Drawer menu item in the Window menu.

1. *Open the Reports disk window by double-clicking on the disk icon.*

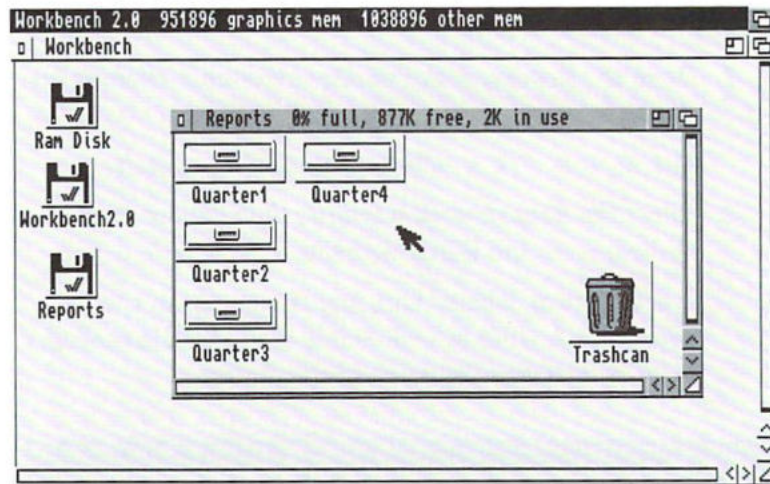
The Reports disk window appears on the screen.

2. *Select the Reports disk window, then choose New Drawer from the Window menu.*

A new drawer, labeled Unnamed1, appears in the Reports disk window. A Rename requester also appears to let you change the name of the drawer.



Follow step 2 to create each additional drawer. Your window would look like this:



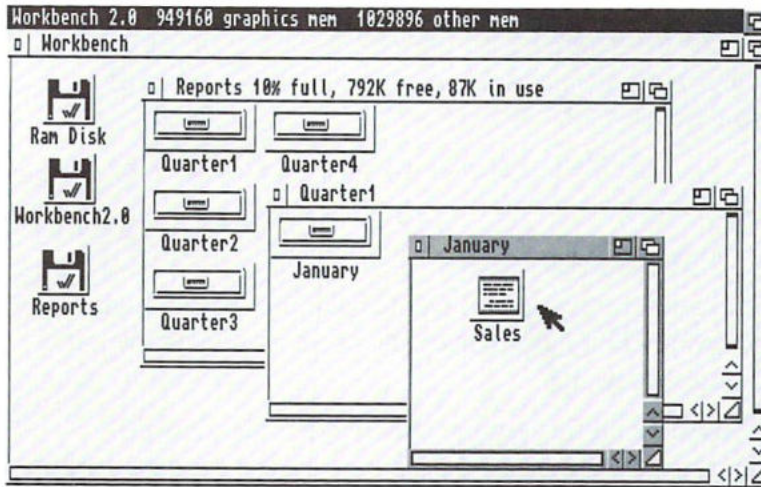
You can store files in any of these drawers, but you might also want to create some drawers within those drawers corresponding to the different months of the quarter.

For example, within the Quarter1 drawer, you might want drawers for January, February, and March. Again, you can use the New Drawer menu item.

1. *Open the Quarter1 window by double-clicking on its drawer icon.*
2. *Select New Drawer from the Window menu.*

A drawer icon labeled Unnamed1 will appear in the Quarter1 window. A Rename requester also appears so you can change the name of the drawer. For this example, name the drawer January.

Create two additional drawers for February and March in the same way. When you wanted to save the Sales report for January, you would put it in the January drawer within the Quarter1 drawer.



Paths

When a program asks you for the name of a file, you must specify the complete **path** to the file. The path specifies the disk name, or location, and all of the drawers that lead to the specified file.

The manner in which you refer to files varies from program to program. Some programs may provide separate boxes (called text gadgets) in which you can enter the disk name, any drawer names, and the filename. However, sometimes you may need to enter the complete path on one line. To correctly cite the path, you must type:

1. The name of the disk followed by a colon. For instance,
Reports:

You can also substitute the disk's volume name with the disk drive name — DF0:, DF1:, or DF2:. However, if you do this, be sure the correct disk is in the drive that you specify.

2. To put a file in the disk window, not in a drawer, specify the filename after the colon. For instance,

Reports:Sales

The icon for the Sales file would be in the Reports disk window.

3. To put the file in a drawer, you must first specify the drawer name, followed by a slash, then the filename:

Reports:Quarter1/Sales

The icon for the Sales file would be in the Quarter1 window. You would first have to open the Reports window, then the Quarter1 drawer.

4. However, in this example, there is another drawer — January. To put a file in the January drawer, you must specify each drawer, followed by a slash, then the filename:

Reports:Quarter1/January/Sales

The icon for the Sales file would be in the January window. You would first have to open the Reports window, the Quarter1 window, then the January drawer.

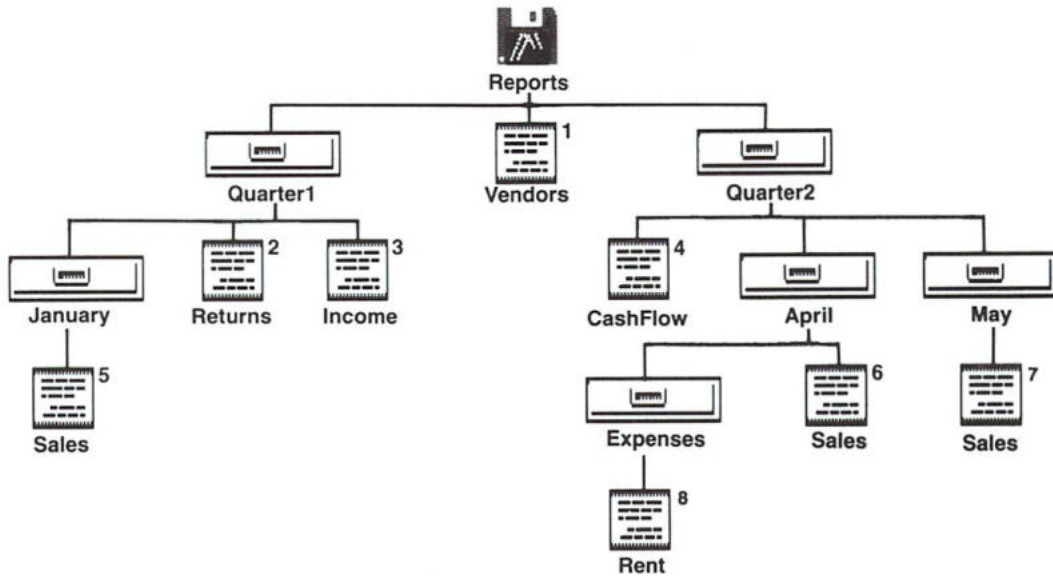
The chart on page 1-53 gives several examples of how to determine path names.

Quick Review

To correctly specify a path, you must type:

1. The disk name or drive name followed by a colon, such as Reports: or DF0:.
2. The complete sequence of drawer names. Each drawer name must be followed by a slash.
3. The filename.

Sample Path Chart



The chart above illustrates a typical disk arrangement. If you were to open the Reports disk icon, you would see the Quarter1 and Quarter2 drawers and the Vendors file in the Reports disk window. If you were to open the Quarter1 drawer, you would see the January drawer and the Returns and Income files in the Quarter1 drawer window. Open the Quarter2 drawer, and you would see the April and May drawers and the CashFlow file, and so on.

It is possible to have several files on one disk with the same name. For instance, in this example there are three files called Sales. However, each Sales file is in a different drawer. So long as the correct path to the file is specified, you do not have to worry about replacing one Sales file with a different Sales file.

The list below shows the correct path to each of the files in the chart:

- 1 Reports:Vendors
- 2 Reports:Quarter1/Returns
- 3 Reports:Quarter1/Income
- 4 Reports:Quarter2/CashFlow
- 5 Reports:Quarter1/January/Sales
- 6 Reports:Quarter2/April/Sales
- 7 Reports:Quarter2/May/Sales
- 8 Reports:Quarter2/April/Expenses/Rent

Naming Files

When choosing names for your files and drawers there are a few rules you must adhere to:

- A filename can be up to 30 characters long.
- Colons (:) and slashes (/) are not allowed within the name. They are reserved to separate disk names and drawer names when specifying a path to a file.
- Beware of using spaces before or after filenames as they are hard to discern on the screen and can cause a lot of confusion.
- Uppercase and lowercase differences between file names are not recognized. This is known as *case-indifference*. However, the Amiga will use the case you specified when displaying the filename.
- You cannot have two files with the same name within the same drawer. If you already have a file named Sales within the Quarter1 drawer, you cannot create a second file also named Sales. The Amiga will replace the original Sales file with the new file of the same name.
- You can have two files with the same name in different drawers. For instance, you could have files named Sales both in the January drawer of Quarter1 and the February drawer. As long as the paths to the files are different, you will not have a problem.

Chapter 2. Basic Operations

In the previous chapter, you learned the basics about using your Amiga, such as using the mouse to move the pointer, using menus, and making backup copies of disks. This chapter reviews what you learned in the tutorial and presents some more detailed information explaining:

- the theory behind the Workbench system
- additional mouse techniques
- how to respond to requesters
- the features of the Workbench screen
- special features of windows
- standard gadgets used on the Amiga
- the different types of icons
- the four Workbench menus

If you were following the steps outlined in the tutorial, you should have rebooted your Amiga with the backup copy of your Workbench2.0 disk. *If you are a new Amiga user and have not read Chapter 1, you should do so before starting this chapter.*

If you have a hard disk system, you do not need to boot with your working disks. Reboot your Amiga by pressing Ctrl, left Amiga, and right Amiga.

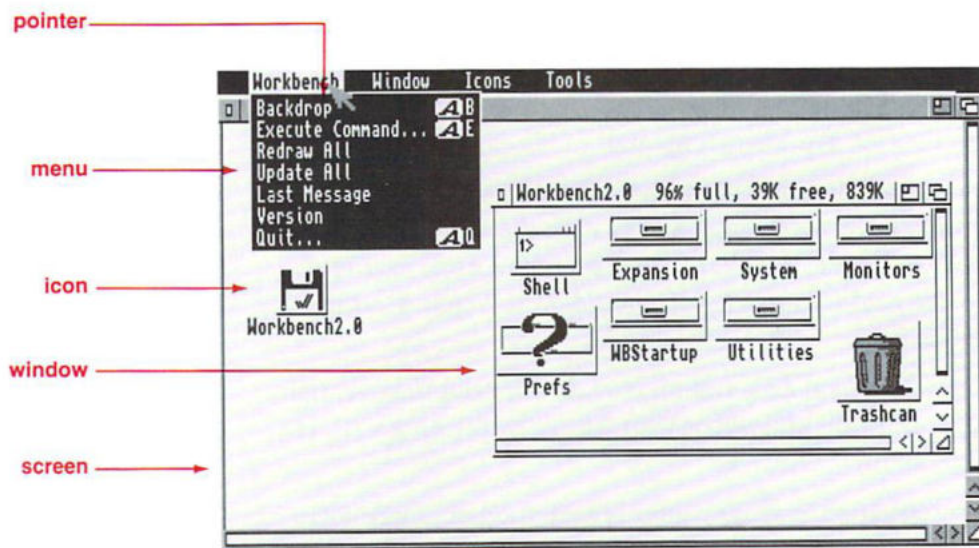


The information in this chapter is presented in independent, easy-to-reference sections. Each section contains both explanatory text and, in most cases, a step-by-step demonstration. For instance, if you need to know about a particular gadget in a window, you can look in the "Windows" section where there is a subheading explaining that gadget.

When you have finished with this chapter, you will be ready to use the programs included on the Workbench2.0 and Extras2.0 disks.

The Workbench System

The Amiga works with a screen/window/icon/menu/mouse/pointer system that is available to all applications. This system is known as the Workbench. Below is a brief explanation of each of these elements:



screen An area of the display with the same video attributes. This includes the number of pixels on each line, the number of colors, and the color palette. Screens are as big as the display area (sometimes larger). Several screens can be open at once, but only one screen at a time can accept information.

- window A rectangular area on the screen that can accept or display information. Several windows can be open on a screen at once. Many programs are run in windows. For instance, you can have a word processing window and a spreadsheet window open at the same time on the Workbench screen.
- icon A small picture that represents disks, drawers, files or programs stored on floppy disks, hard disks, or on other storage devices.
- menu A list of options from which you can choose a specific operation, such as copying a file, renaming an icon, or organizing the contents of a window.
- mouse A small mechanical device used to communicate with the Amiga. There are two buttons on the mouse that allow you to select icons to work with and choose options from menus.
- pointer An image that you can move across the screen by moving the mouse. When you move the pointer over an icon or to certain areas of the screen, you can then use the mouse buttons to send a message to the Amiga.

All of these elements combine to provide the icon-based environment that you use to interact with your Amiga. It is this environment that is known as the Workbench.

You can compare the Amiga Workbench to a carpenter's workbench. When a carpenter is building a cabinet, he spreads his tools and his materials out upon his workbench and then proceeds to build the cabinet upon that same surface.

In a similar way, the Amiga Workbench provides you with a computerized work surface. Think of the pointer, menus and windows as your tools and of the programs on the Workbench2.0 disk, or any other application software, as your

materials. You use your tools and materials on the Amiga Workbench to build or create a file, whether it's a text file from a word processing program, an animation file from a video program, or a data file from a spreadsheet program.

The main visual component of the Workbench is the Workbench screen. Even if you are using a commercial application, chances are that the first screen you see will be the Workbench screen. Sometimes an application will start with a specialized screen, but the Workbench screen is usually present somewhere in the background.

The floppy disk that contains the basic Amiga software is called the Workbench2.0 disk, and when you open this disk the window that appears is called Workbench2.0. This results in several elements sharing the Workbench name, but they are all parts of the Workbench environment.



If you have a hard disk, the Amiga software is on the partition called System2.0. It contains the exact same software as the Workbench2.0 and Extras2.0 floppy disks.

There is an alternative, keyboard-based method for communicating with the Amiga. You can do this through the Shell program, which is explained in full detail in the AmigaDOS section of this manual. Many operations can be performed both through the Workbench and through the Shell.

Mouse Techniques

The mouse lets you communicate with your Amiga by moving and positioning a pointer then pressing one of the mouse buttons. Even though there are only two buttons, there are many operations that you can perform.

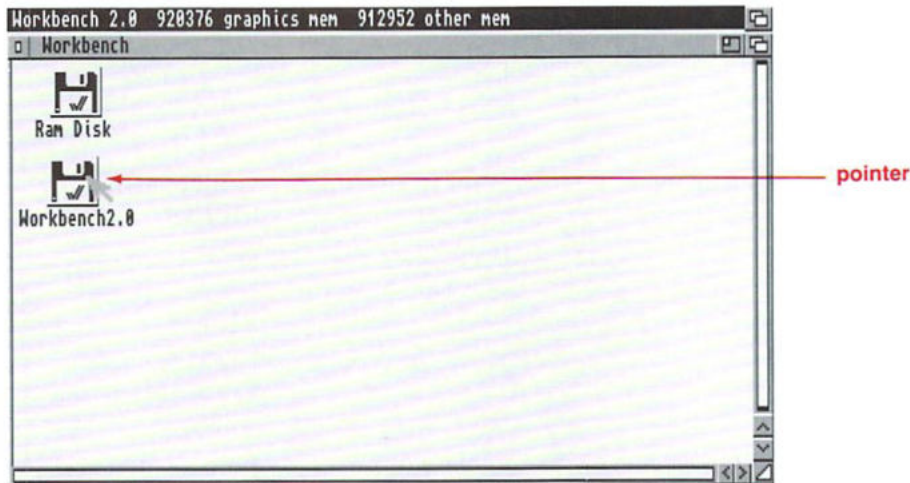
When an instruction tells you to click the mouse button, it means to press and release the button. **Holding down** the mouse button means to press the button until you are told to release it.

To point to an icon, use the mouse to move the pointer so that its tip is positioned over the item, as shown below:

Click = press and release

Hold down = press continually

Point = move pointer tip so that it is over an object on the display



In Chapter 1, the word right or left was specified in instructions telling you to press a mouse button. Throughout the rest of the manual, however, the buttons will be referred to by their correct names. The left button is the selection button, and the right button is the menu button.

Using the Amiga without a Mouse

All mouse actions can be accomplished with the keyboard. You can use the keyboard to move the pointer, select icons, and choose menu items. (A description of the keyboard can be found in your *Introducing the Amiga* manual.) The chart below outlines the keyboard methods:

Operation	Mouse Method	Keyboard Method
Moving the pointer	Move the mouse.	Hold down an Amiga key and a cursor key. (Hold down Shift to move the pointer faster.)
Selecting an icon, window or screen	Point to an icon, window or screen.	Point to an icon, window, or screen.
	Click the selection button.	Hold down left Amiga and left Alt.
Dragging	Point to an icon or title bar.	Point to an icon or title bar.
	Hold down the selection button.	Hold down left Amiga and left Alt.
	Move the mouse.	Use the cursor keys to move the pointer.
	Release the selection button.	Release all keys.

Operation	Mouse Method	Keyboard Method
Drag selection	Hold down the selection button.	Hold down left Amiga and left Alt.
	Move the mouse to draw a box around the icons.	Use the cursor keys to move the pointer to draw a box around the icons.
	Release the selection button.	When the box is drawn, release all keys.
Choosing a menu item	Hold down the menu button to display menus.	Hold down right Amiga and right Alt to display menus.
	Point to a menu heading.	Keep holding down keys, and use the cursor keys to point to a menu heading.
	Point to a menu item.	Keep holding down keys, and use the cursor keys to point to a menu item.
	Release menu button.	Release all keys.
Cancelling	Hold down menu button while selection button is pressed.	Hold down right Amiga and right Alt while left Amiga and left Alt are pressed.
The keyboard equivalent to clicking the left mouse button is to press left Amiga and left Alt.		
The keyboard equivalent to clicking the right mouse button is to press right Amiga and right Alt.		

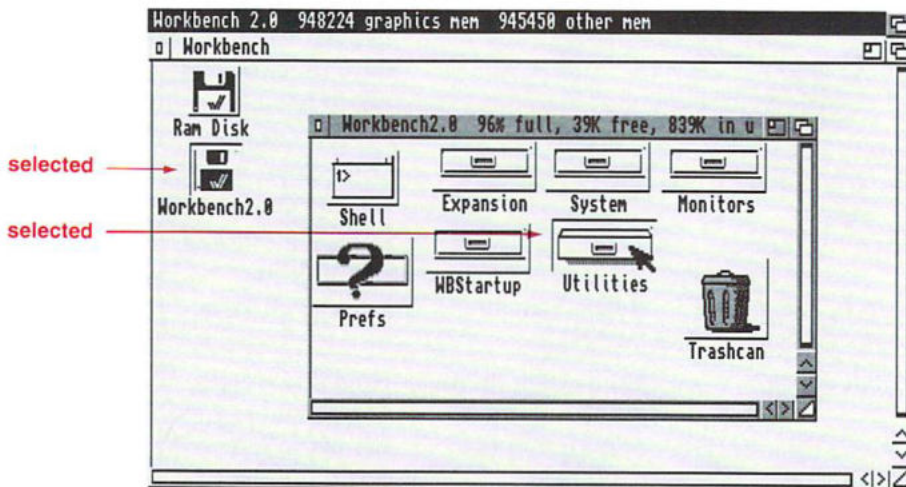
The Selection Button

The left mouse button is the selection button. With this button, you choose, or select, the icons, windows, or screens that you want to use. You can also use the selection button to move, or drag, items around the screen.

Selecting

You need to select an icon, window, or screen before you can work with it. For instance, when an icon is selected, you can make a copy of it, change its name, and even delete it.

All icons are surrounded by a box. When an icon is not selected, the box appears raised above the screen or window surface. When an icon is selected, the box appears to sink into the screen or window surface.



When you select an icon, the box appears to sink into the screen or window surface. Some icons may change color or shape when selected. Drawer icons may change from a closed drawer to an open drawer.

To select an icon:

1. *Point to the icon.*

Make sure the pointer tip is within the icon's box.

2. *Click the selection button.*

The icon will change to show that it is selected.

If you click the selection button while the pointer is elsewhere on the screen or window, the icon will no longer be selected and will return to its original appearance.

To select a screen or window:

1. *Click the selection button while the pointer is inside the screen or window, but not over an icon.*

When a window is selected, the frame surrounding the window changes color. When a Workbench window is selected, the frame of that window is highlighted and the amounts of available memory are displayed across the top of the screen.

Selecting Multiple Icons

At times you may want to select several icons at once. When multiple icons are selected, you can treat them as a single entity. You can delete, move, or copy the entire group in one operation.

There are three ways to select multiple icons: **drag selection**, **extended selection**, and the Select Contents menu item, explained on page 2-62.

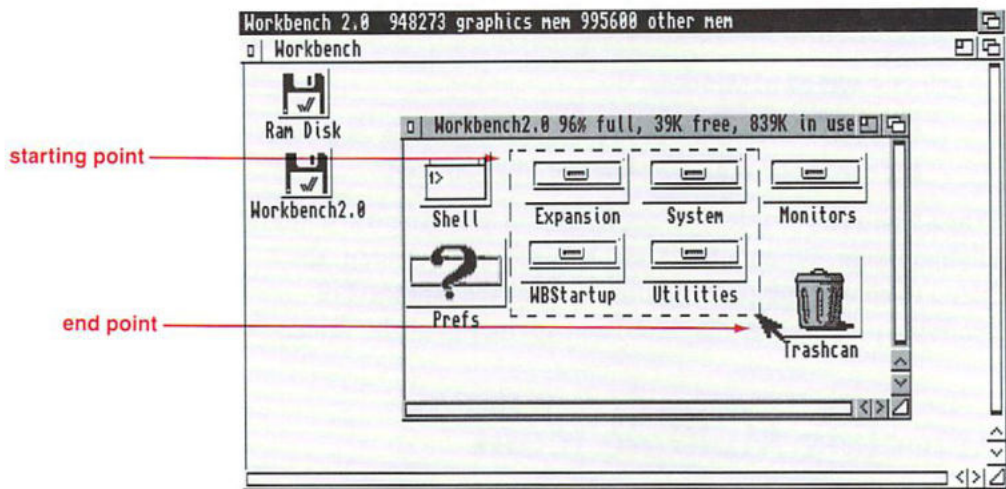
Drag selection is a way to select several icons at once by using the mouse to draw a box around them. To do this:

1. *Move the pointer just outside of the outermost icon you want to include in the box.*

Do not let the pointer touch any icons.

2. *Hold down the selection button, and move the mouse.*

As you move the mouse, a dotted box will be drawn.



3. *When the box encloses the icons that you want to select, release the mouse button.*

All of the icons inside the dotted box will be selected.

Extended selection is useful when the icons you want to select are not in a group that you can enclose in a box. To use extended selection:

1. *Select the first icon.*
2. *Hold down Shift.*
3. *Select the other icons.*

While holding down Shift, point to each icon and click the selection button.

4. *Release Shift.*

Each icon you have clicked on will be selected.

Double-clicking

Double-clicking means clicking the selection button twice in rapid succession. When you point to an icon and double-click the selection button, a window appears or a program is started.

You can adjust the time allotted for a double-click with the Input Editor, explained on page 3-7.

Dragging

Dragging is the act of moving an icon, window, or screen.

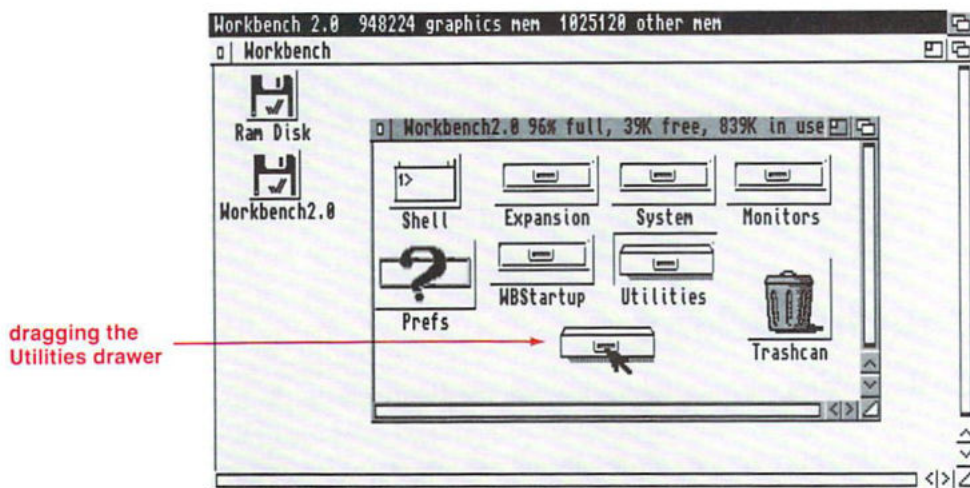
Dragging an Icon

You can move an icon into another window by dragging it out of the original window and into a new window. You can also copy and delete icons by dragging them to certain areas on the screen or in a window. (This is explained in the "Icons" section.)

To drag an icon:

1. *Point to the icon.*
2. *Hold down the selection button, and move the mouse.*

A copy of the icon will move with the pointer.



3. *Release the selection button when the icon is positioned where you want it.*

If you have selected several icons, you can drag all of the icons at once. Hold down Shift, point to one of the icons, hold down the selection button, and move the mouse. All the selected icons will move as you move the mouse.

Dragging a Window

When you have several open windows on the Workbench screen, they may overlap each other. You can move the windows around by dragging them. This helps you see the information presented in all the windows.

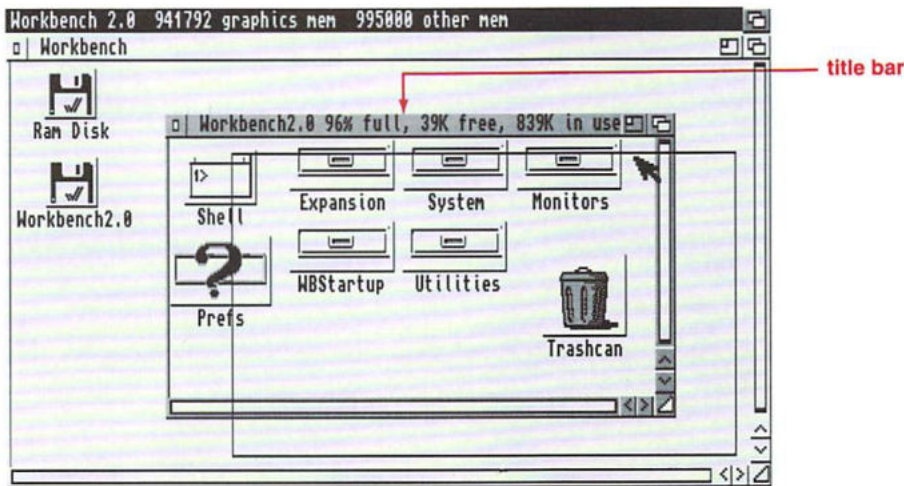
To drag a window:

1. *Point to the top border of the window, but make sure the pointer is not over any of the square gadgets at either corner of the border.*

This area is known as the window's title bar.

2. *Hold down the selection button, and move the mouse.*

An outline of the window appears and moves across the screen.



3. *Drag the outline to where you want the window, then release the selection button.*

When you release the selection button, the window appears in the new location.

Dragging a Screen

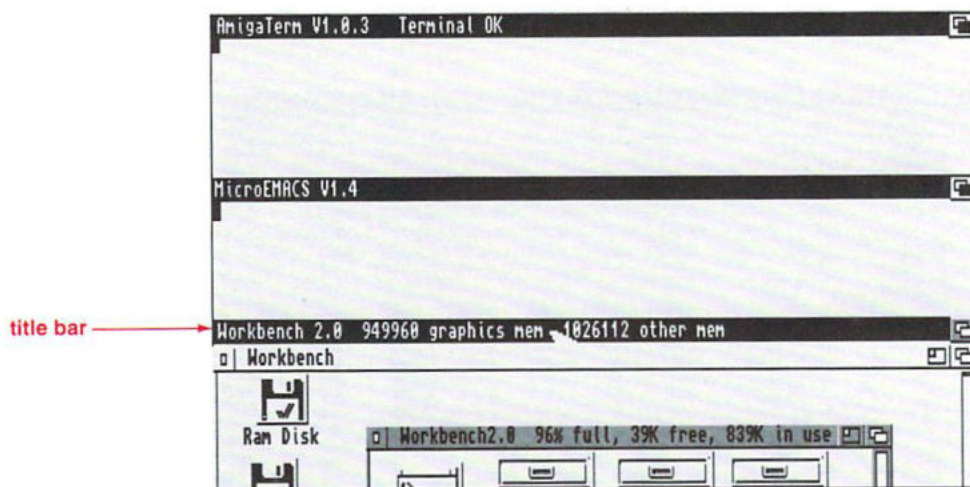
With the Amiga, it is possible to have more than one screen open at a time. For instance, you can have your Workbench screen, a terminal program screen that lets you communicate with other computers, and a text editor open at the same time. You can see parts of each screen by dragging them.

To drag a screen:

1. *Point to the screen's title bar.*

This is the area across the top edge of the screen.

2. *Hold down the selection button.*
3. *Move the mouse down.*



To expose a screen, you can only drag the front screen down, not up. However, if a screen is larger than the monitor's display area, you can drag it up or down or side-to-side so that you can see all areas of the screen.

An alternative method for dragging a screen is to:

1. *Place the pointer anywhere on the screen.*
2. *Hold down left Amiga.*
3. *Hold down the selection button and move the mouse up or down.*

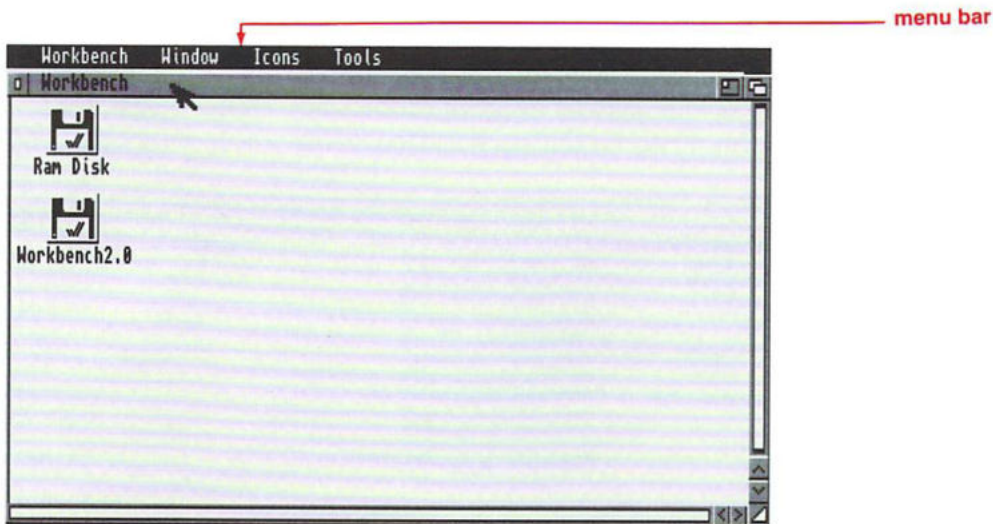
The Menu Button

The right mouse button is the menu button and is used to display menus and to choose items from them.

You can also use the menu button to cancel an operation that is being performed with the selection button, such as drag selection. Cancelling is described at the end of this section.

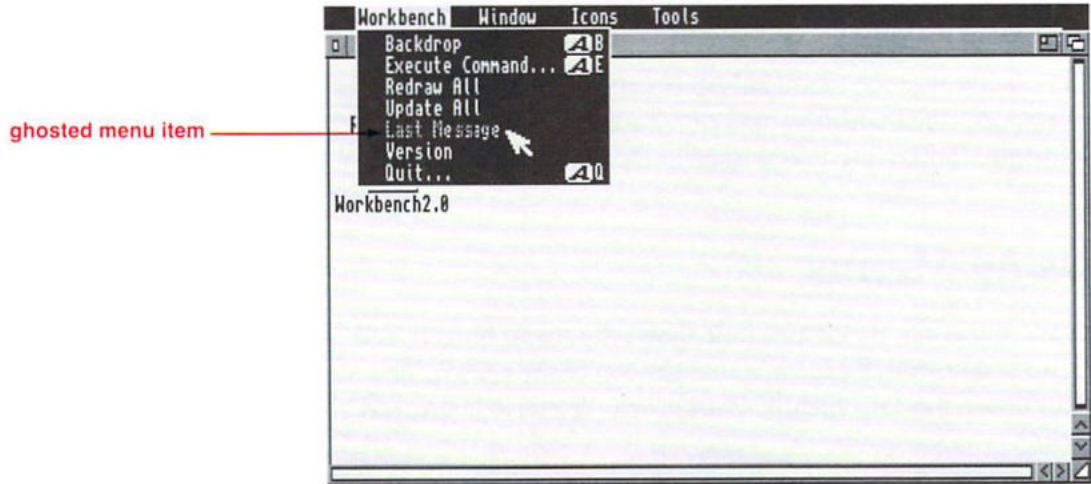
Using Menus

To see the available menus, hold down the menu button, and menu headings will appear across the top of the screen. The Workbench has four menus: Workbench, Window, Icons, and Tools.

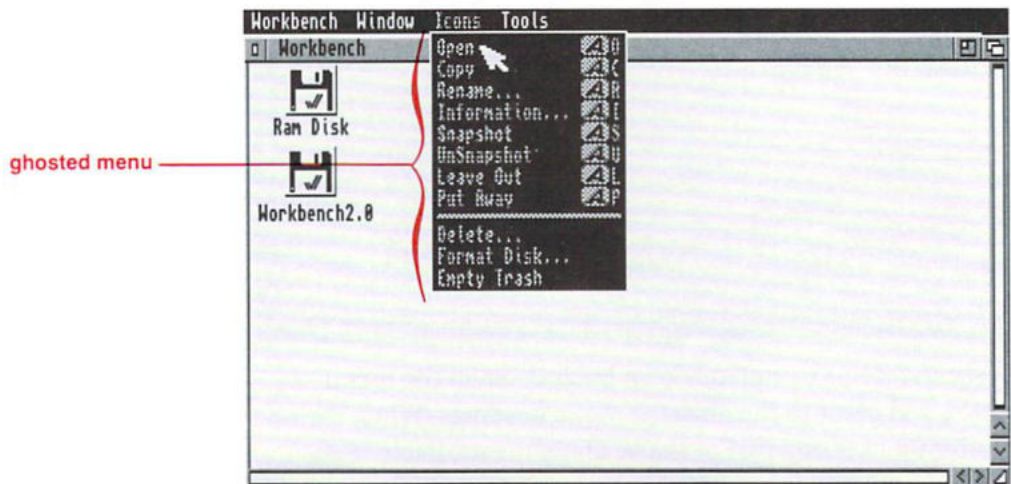


To see the items in a menu, keep holding down the menu button and point to the different menu headings. When the pointer touches any part of the heading, the available items will be listed beneath the heading.

Not all menu items are available at all times. The unavailable items are ghosted (not displayed clearly) and will not be highlighted when the pointer passes over them.



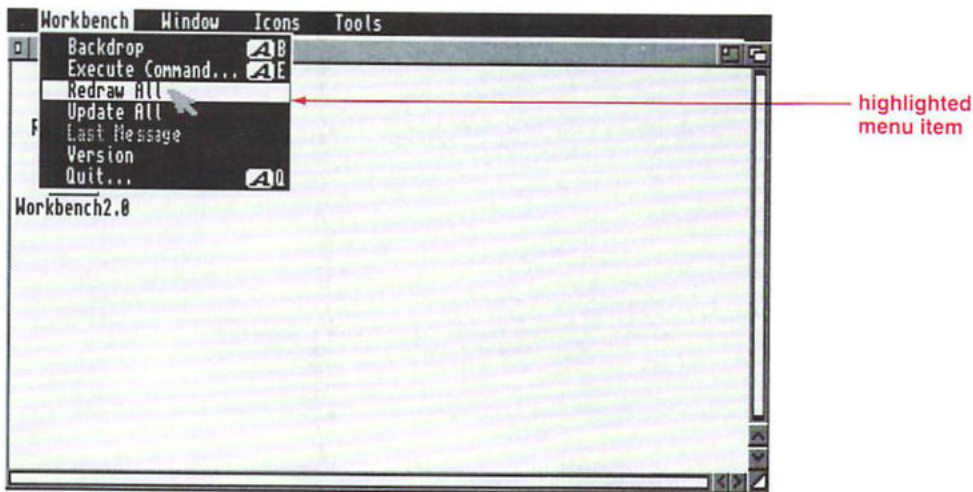
Sometimes menu items are ghosted because something on the screen needs to be selected. For instance, all the items in the Icons menu are ghosted if an icon on the screen is not selected.



To choose a menu item:

1. *Continue to hold down the menu button.*
2. *Move the pointer down the menu.*

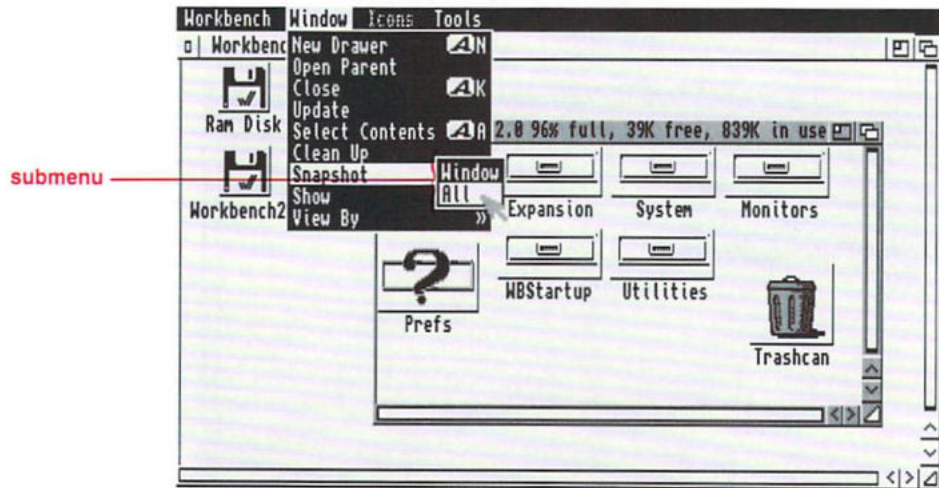
As you point to each available menu item, it will be highlighted.



3. *When the menu item you want to choose is highlighted, release the menu button.*

Some menu items may have **submenus**. Submenus are additional options that appear to the right of the menu item. (If a menu item has a submenu there will be a \gg after the item name.) You must choose an option from the submenu list to use the menu item.

For instance, when you point to the Snapshot menu item in the Windows menu, a submenu appears. You must choose either Window or All in order to use Snapshot. (The two options are explained in the "Snapshot" section on page 2-63.)



To choose an option from a submenu:

1. *Point to the main menu item.*

The menu item will be highlighted, and the submenu will appear to the right of the menu item.

2. *Keep holding down the menu button, move the pointer to the first item in the submenu, then move down the list to the item you want to choose.*

The submenu item will be highlighted.

3. *Release the menu button.*

Cancelling

You can cancel an operation being performed with the selection button by clicking the menu button *while still holding down the selection button*. The following operations can be cancelled: selecting, dragging, and drag selection. Some examples of cancelling follow.

If you have several icons selected, and you want to cancel the selection of all of them, click on an empty area of the screen.

To cancel the selection of one icon:

1. *Point to the icon.*
2. *Hold down Shift and the selection button, then click the menu button.*

If you are dragging an icon or a window across the screen, and you decide you want to leave it in its original location, click the menu button before releasing the selection button. The operation will be cancelled, and the icon or window will remain in its original position.

You can cancel drag selection in the same way. If you are selecting several icons, click the menu button *without releasing the selection button* to cancel the operation. None of the icons will be selected.

Requesters

Before you begin to examine the individual aspects of the Workbench system, take a few minutes to learn about requesters.

A requester is a small window opened by a program, like Workbench, when it needs a response from you. You will come across requesters repeatedly as you start to use the various parts of the Workbench, like menus, windows and icons. When a requester window appears, it is immediately brought to the front of the display and is automatically selected.

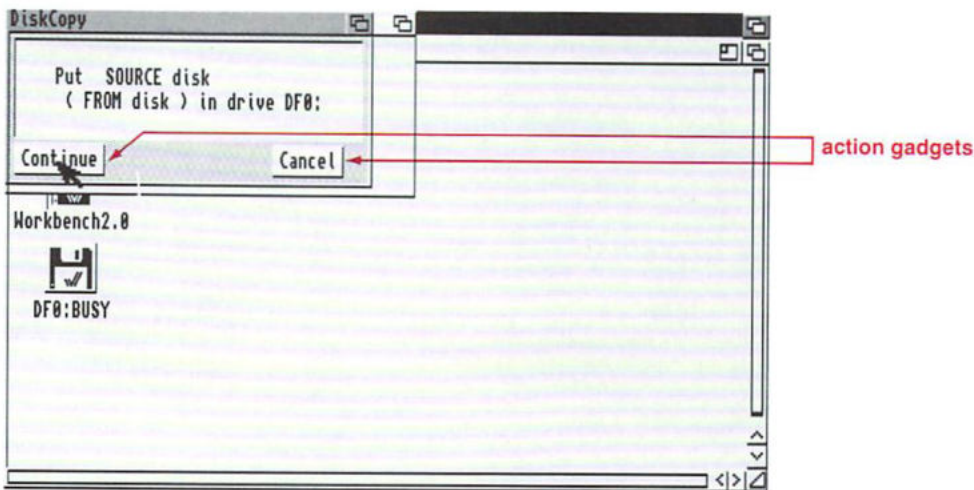
Requesters are often the result of a menu choice. Several of the menu items are followed by three dots (. . .) to indicate that they generate requesters.



A requester will always contain text explaining what you must do. Be sure to read the text in the requester before selecting a gadget or entering text.

Action Requester

Some requesters ask you to choose between two options. For instance, it may ask you if you are sure you want to proceed with an operation or if you want to cancel the operation.



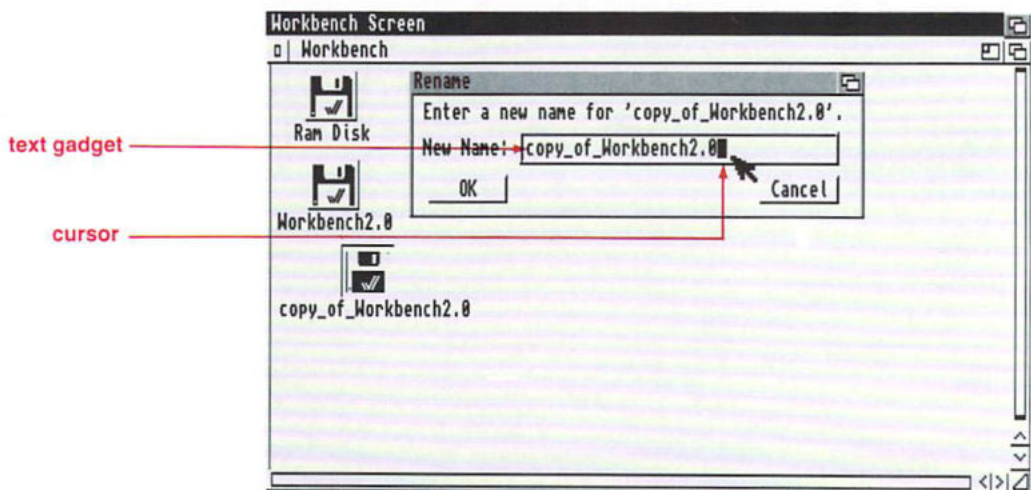
These requesters contain two **action gadgets**. One gadget lets you proceed with an operation. It is usually labeled OK, Continue, or Retry. The other gadget is a Cancel gadget and stops the operation without performing any action. To choose one of the options, select the appropriate gadget.

Keyboard Shortcut: To select the gadget that lets you proceed (OK, Continue, or Retry), press left Amiga-V. To select the Cancel gadget, press left Amiga-B.

You can change these keys with the IControl editor explained in Chapter 3.

Text Requester

Another type of requester is one that asks you to enter text. This type of requester contains a **text gadget**, a rectangular box that allows you to enter text. For instance, the following requester appears when you choose the Rename menu item.



When the requester appears the text gadget is automatically selected. When you type at the keyboard, the text appears to the left of the **cursor** (the small, highlighted box inside of the text gadget).

If you click the selection button while the pointer is somewhere else on the screen, the requester may no longer be selected. To enter your text, you will have to move the pointer inside of the text gadget, and click the selection button.

Sometimes text gadgets will contain information that needs to be changed. For instance, when you choose the Rename menu item, the text gadget may contain the current name of the icon. Some shortcuts for editing text within a text gadget are listed below:

Del	Erases the character highlighted by the cursor.
Backspace	Erases the character to the left of the cursor.
right Amiga-X	Erases all the text in the gadget.
right Amiga-Q	Retrieves what was in the gadget before the text was changed.
Shift-left cursor	Moves cursor to the beginning of the line.
Shift-right cursor	Moves cursor to the end of the line.
Shift-Del	Erases the character highlighted by the cursor and all characters to the right of the cursor.
Shift-Backspace	Erases all the characters to the left of the cursor.

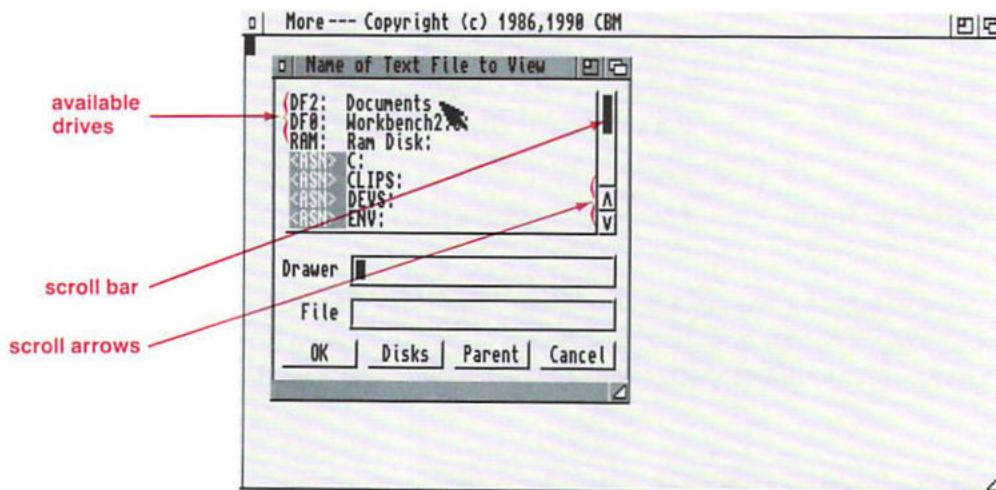
When the text in the gadget is correct, press Return. (With some programs, you may also need to select an action gadget.) The requester will disappear, and the action will be carried out.

File Requester

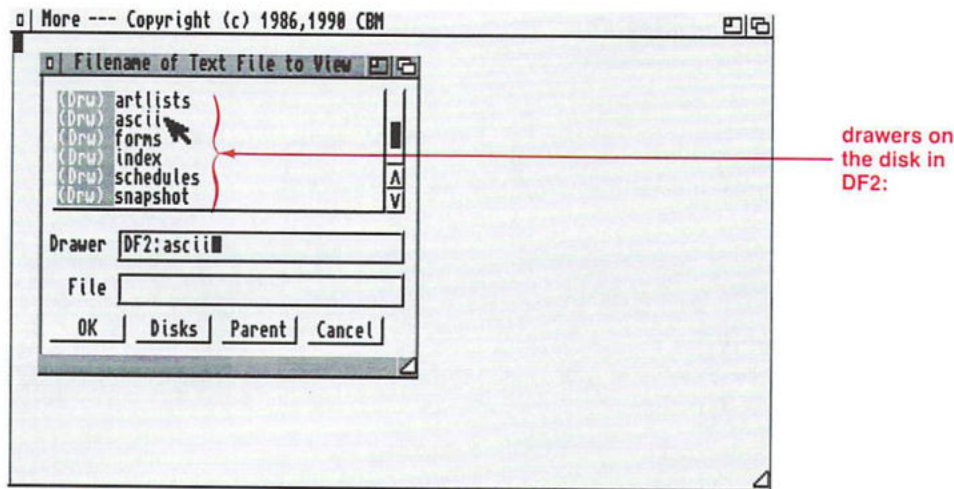
Another type of requester is one that allows you to enter the name of a file. These requesters usually appear so you can specify a file from which to read or to save information.

For instance, the More program in the Utilities drawer allows you to display the contents of text files. When you open More, a file requester containing a list of the files on the Workbench disk appears. To read through the list, drag the scroll bar up or down or select the scroll arrows. If the file you want to use is in a different drawer or on another disk, the gadgets in the requester allow you to look for that file.

Select the Disks gadget, and a list of available floppy drives, hard disk partitions and assigned volumes will be displayed. (Assigned volumes are explained later in this manual.)



To list the available files and drawers on a disk, point to the disk's name and click the selection button. The display will change to list the files and drawers on that disk.



If the file is in a drawer, point to the name of the drawer, click the selection button, and a list of files in the drawer appears. When the correct filename is displayed, point to the filename and click the selection button.

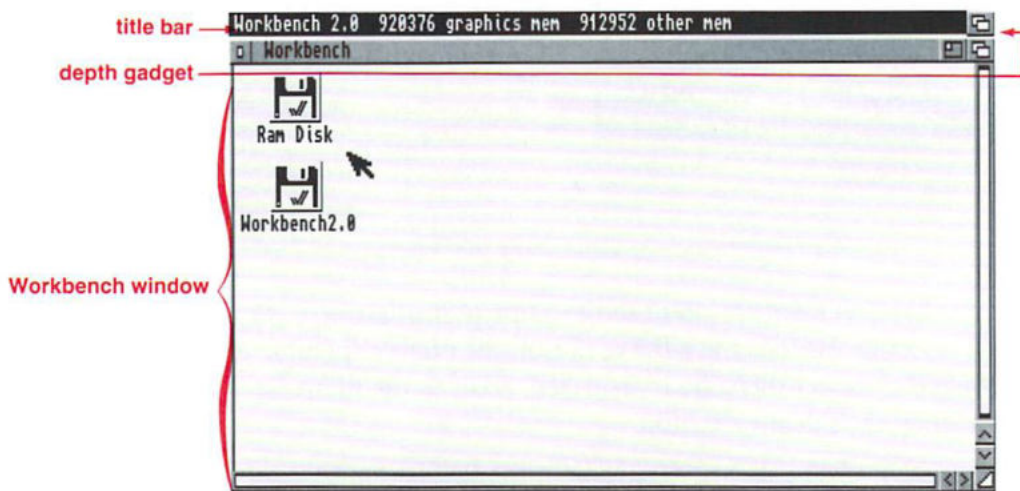
When you select a disk or drawer from the list, its name appears in the Drawer text gadget. When you select a filename, it appears in the File gadget. You can also type the correct disk and drawer names directly into the text gadgets. Click inside the text gadget, and a cursor will appear. You can then type the proper names.

The Parent gadget returns you to the **parent** drawer of the currently displayed list of files. The parent drawer is the drawer that is one level above the currently shown drawer. For instance, if you're looking at a list of files in the Utilities drawer, selecting the Parent gadget will return you to the list of drawers and files on the Workbench2.0 disk.

Once the correct filename is displayed, select the OK gadget. If you change your mind and want to exit the requester, select the Cancel gadget.

The Workbench Screen

The Workbench screen provides the background for your work. Icons and windows appear on this screen.



Title Bar

The top border of the screen is known as the title bar. When a Workbench window is selected, the title bar shows the name of the screen, as well as how much memory is available. Graphics mem refers to available **Chip RAM**. The amount of Chip RAM in your system determines how much memory is available for graphics and digitized sounds. Other mem refers to all other available RAM including any expansion, or **Fast**, RAM used by the system.

When you hold down the menu button, the menu bar is displayed. The menu bar lists the menu headings. Available menu headings are shown clearly, while unavailable menu headings are ghosted.

Workbench Window

When you boot your Amiga, the Workbench window fills the Workbench screen. This window contains icons for any hard disks attached to your Amiga, floppy disks that are in any of the drives, and the Ram Disk. Depending on your system's configuration, it is possible that there may be other icons in the window.

Although the Workbench window looks and acts like a window, it is an essential part of the Workbench screen. When the Workbench window is selected, the Workbench screen is also selected.

Moving the Workbench Screen

Since the Amiga is multitasking, it is possible to have multiple screens open at the same time. For instance, you can be running a graphics program on the Workbench screen and a terminal program connecting you to an electronic bulletin board on another screen. In this case, you will have to move the Workbench screen in order to see the other screen. There are several ways to move the Workbench screen.

One way to drag the screen is by pointing to the title bar, holding down the selection button, and moving the mouse. Another way is to hold down left Amiga and the selection button, point anywhere on the screen or Workbench window and move the mouse up or down.

The small box in the upper right corner of the Workbench screen is its depth gadget. You can click on this gadget to move the Workbench behind or in front of other screens.

You can also move the Workbench screen with the keyboard. Pressing left Amiga-N moves the Workbench screen *in front of* all other screens. Left Amiga-M moves the front screen *behind* all other screens.

You can customize this action to a key of your choice with the IControl editor explained in Chapter 3.

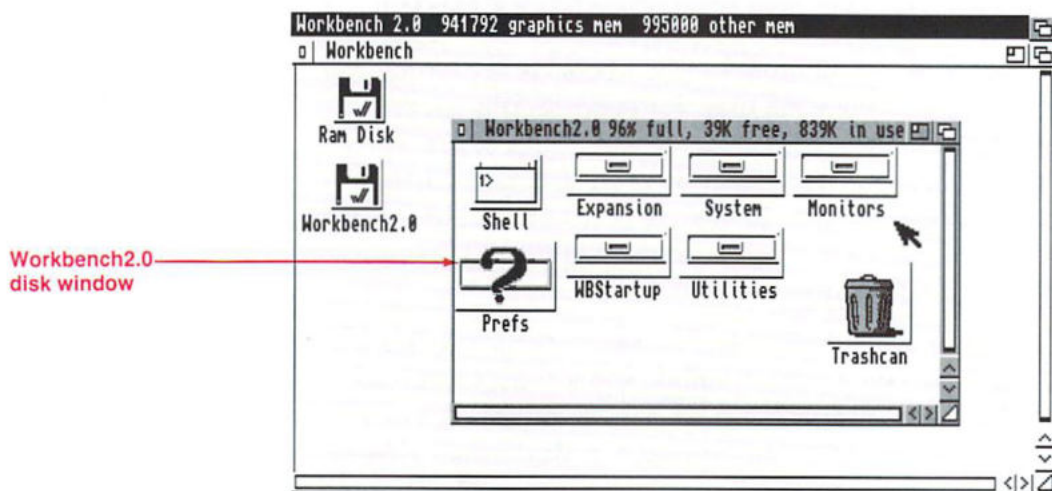
These keys can also be changed with the IControl editor.

Windows

A window is an area of the screen that displays and accepts information. There are many different types of windows. You've already seen the Workbench window that appears when you boot your Amiga. Double-click on the Workbench2.0 disk icon, and a window appears displaying the contents of the Workbench2.0 disk.



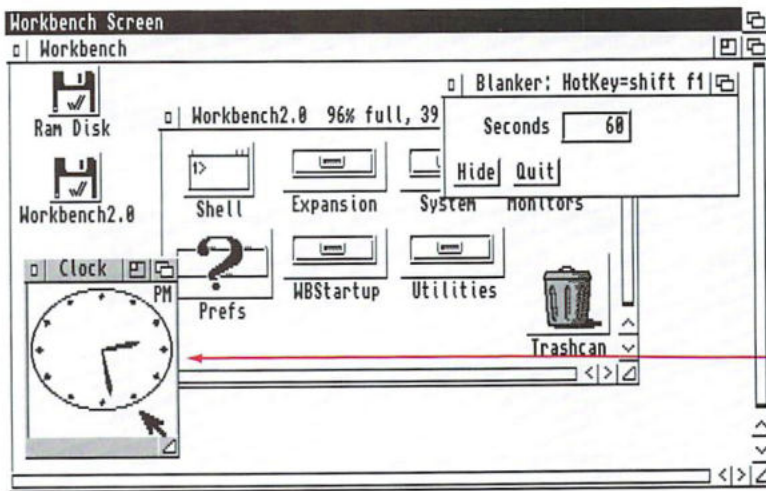
If you have a hard disk, double-click on the System2.0 icon. Your disk window will have more drawers than are pictured here since the System2.0 partition contains the drawers for both the Workbench2.0 and Extras2.0 disks.



Many of the programs on the Workbench2.0 disk create windows when their icons are opened. You may find that much of the commercial software you purchase also opens windows.

New windows open on the front of the screen. While several windows can display information simultaneously, only one window at a time can accept information. This window is known as the selected, or active, window. When a window is selected its border, or frame, is a different color from the other windows on the screen.

Often when a window fills the screen, it is referred to as a screen. Technically, this is incorrect. Although it may fill the screen, it is still a window.



To select a window, point anywhere inside the window or its title bar, and click the selection button. Clicking the selection button while the pointer is outside of the window will deactivate the window. It will no longer be selected.

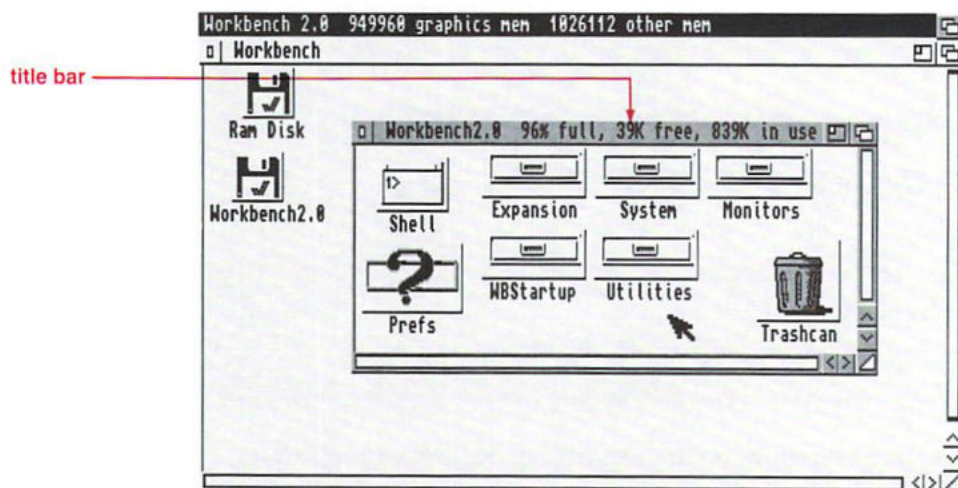
When you have several windows open on one screen, it often results in windows overlapping each other. When you want to look at the contents of a particular window, you may need to move other windows around so that you can see the one you want. Most windows have gadgets to allow you to perform these actions. Gadgets are the boxes and bars in a window's border. Some of the most common gadgets are explained in the following sections.

Different programs may use different gadgets. When a program uses a unique gadget, it is usually explained in the program documentation.

Title Bar

Across the top of each window is a title bar that shows the name of the window. For Workbench windows, the name will be the same as the name of the icon that was opened to create the window.

For instance, when you open the Workbench2.0 disk icon, the window will have a title bar that looks something like this:



*One kilobyte (K) =
1,024 bytes*
*One megabyte (MB) =
1,024 kilobytes*

The information in this title bar identifies how much data is on the disk. It states what percentage of the disk is *full*, how many kilobytes are *free* (available for storage), and the number of kilobytes that are *in use* for existing data. If you have a hard disk system, these values may be expressed in megabytes.

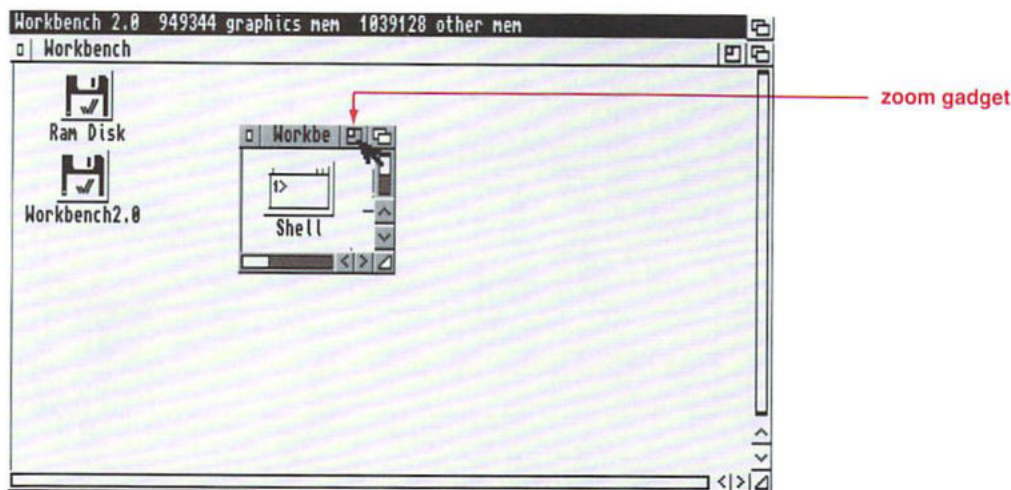
When you open a drawer icon, the title bar only displays the name of the drawer.

As explained earlier, you also use the title bar to drag a window.

Zoom Gadget



Selecting the zoom gadget changes the size of a window. In some cases, as with the Workbench 2.0 disk window, the window becomes smaller.



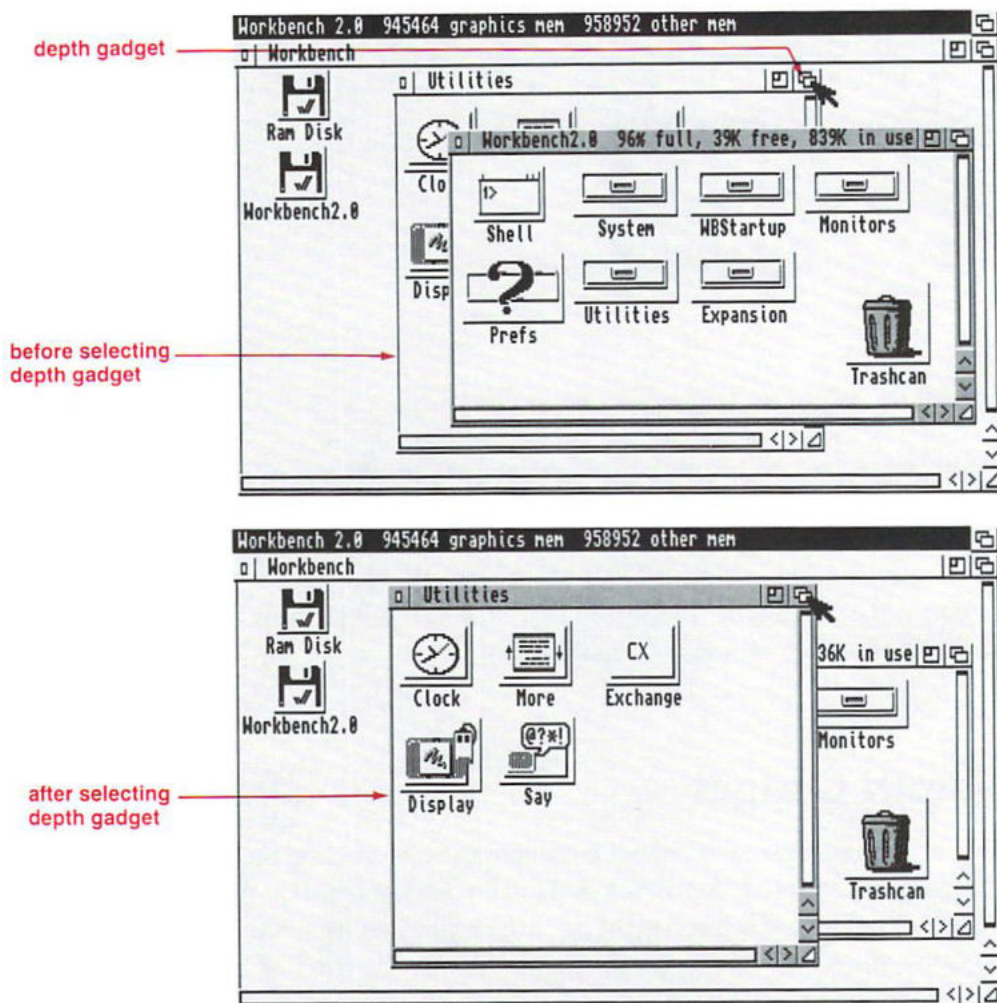
Selecting the zoom gadget a second time returns the window to its previous size and position. If you change the size of a window, the window will return to that new size and position the next time it is zoomed in and back.

Depth Gadget



When you have several windows overlapping each other on the screen, you can move them back and forth with the depth gadget. If a window is front-most on the screen (not obscured by other windows), selecting this gadget pushes that window behind all the other windows.

Selecting the depth gadget on any window other than the front-most window, brings that window to the front. For instance, if you have three windows open on the screen, selecting the depth gadget of the middle window will bring it to the front of the screen.

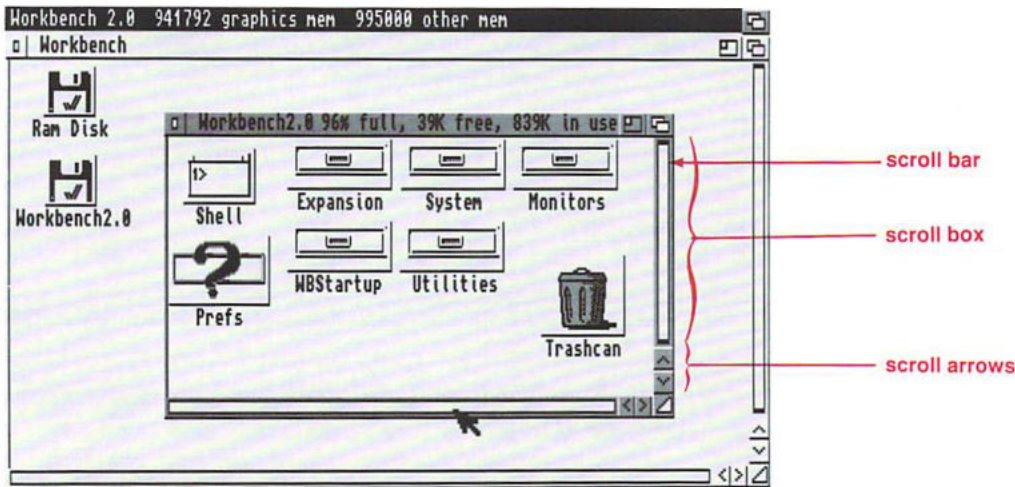


If you hold down Shift and select the depth gadget, the window will move behind all the other windows.

Scroll Gadgets

Sometimes a window is not large enough to show all of its icons. You can see the window's contents without changing its size by scrolling the window. Scrolling refers to moving the viewing area of a window so that you can see unexposed icons.

Most windows have two scroll gadgets, one along the right edge of the window and one along the bottom. The scroll gadget is made up of a scroll box, a scroll bar, and scroll arrows.

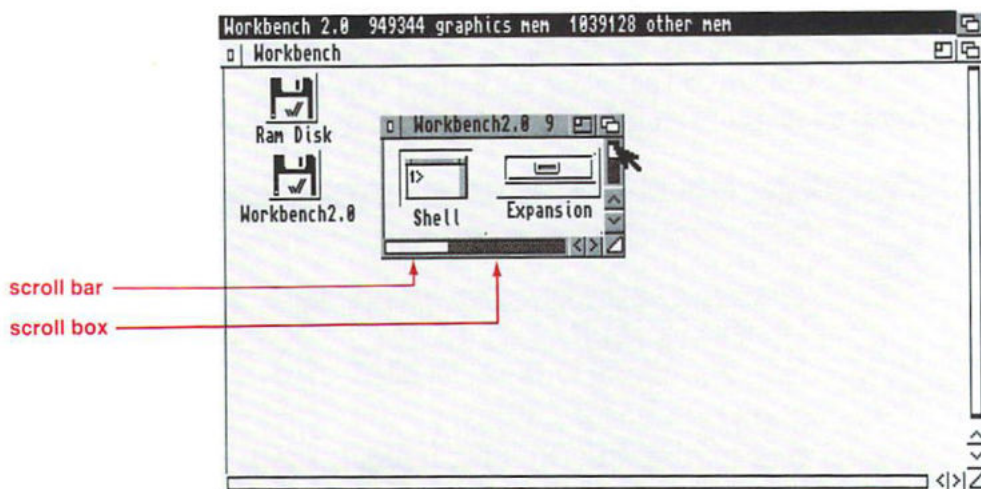


The scroll bar is the highlighted rectangular area inside of the scroll box. The size of the scroll bar indicates how much of the window is visible. If the entire window is visible, the scroll bar fills the entire scroll box. However, if the scroll bar only fills half of the scroll box, only half of the window is visible. By dragging the scroll bar to the empty area of the scroll box, you can see the obscured icons.

The position of the scroll bar reflects which portion of the window is visible. For instance, if the scroll bar is in the upper half of the vertical scroll box, you can see the icons in the top of the window.

To drag a scroll bar:

1. *Point at the scroll bar.*
2. *Hold down the selection button.*



3. *Use the mouse to drag the scroll bar to an empty area of the scroll box.*

The viewing area of the window will move in the same direction as the scroll bar.

Another way to move the scroll bar is to point to an empty area of the scroll box and click the selection button. The scroll bar will move to the area where you have pointed.

You can also use the scroll arrows to scroll the viewing area of a window whether or not all the icons are visible. This can expose empty areas of the window into which icons can be moved.

To use a scroll arrow:

1. *Point to a scroll arrow.*
2. *Click the selection button.*

The viewing area of the window will shift in the direction of the arrow.

To move the contents more quickly, hold down the selection button while pointing to a scroll arrow.

Sizing Gadget

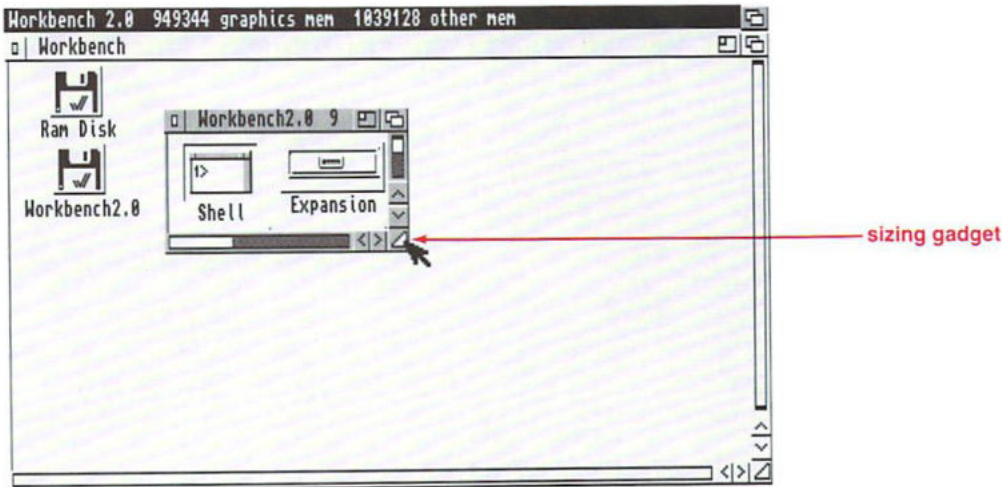


As its name implies, the sizing gadget lets you change the size of the window — making it larger or smaller as needed.

To change the size of a window:

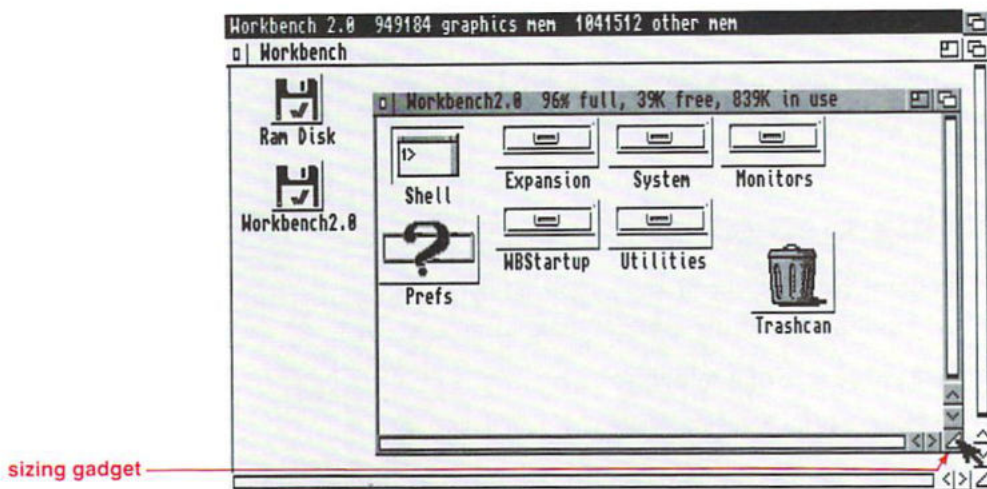
1. *Point to the sizing gadget.*
2. *Hold down the selection button, and move the pointer towards the upper left corner of the screen.*

The window will become smaller.



3. Point to the gadget again, hold down the selection button, and move the pointer towards the lower right corner of the screen.

The window will become larger.



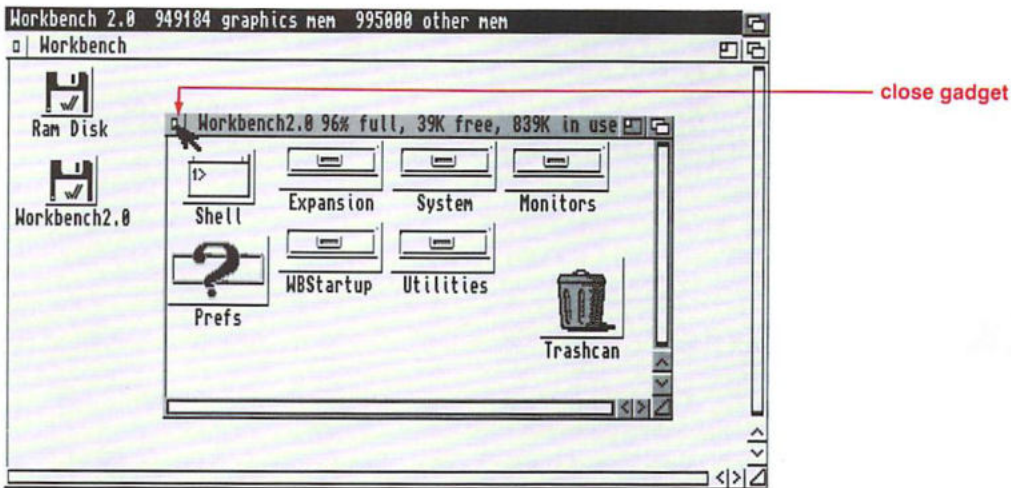
When you have sized the window, release the selection button.

You can cancel the sizing operation by pressing the menu button before releasing the selection button.

Close Gadget



Selecting the close gadget **closes** the window. When you are through working in a window, select the close gadget, and the window will disappear.



If you select the close gadget on the Workbench window, a requester will ask you if you really want to quit the Workbench. If you select OK, all Workbench functions will be closed, including any Shell windows started from an icon. *You cannot close the Workbench if you have any programs running.*

In order to leave a Shell window open, you must start one with the NEWSHELL command. There are two ways to do this:

- 1. Open a Shell from the icon, then type NEWSHELL at the prompt.*
- 2. Use the Execute Command menu item in the Workbench menu.*

Type NEWSHELL in the text gadget.

To get the Workbench back, type LOADWB (load Workbench) at the Shell prompt and press Return.

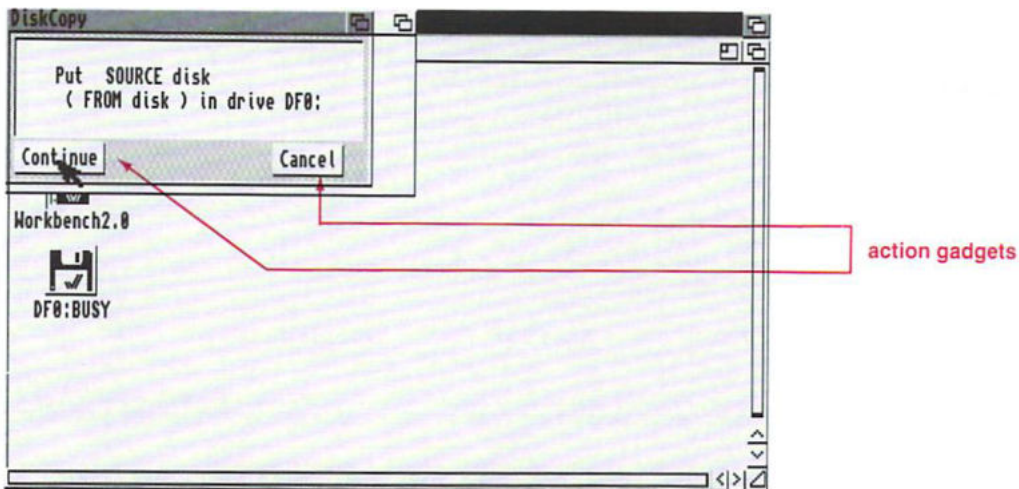
Additional Gadgets

In the “Windows” section, you learned about the various gadgets that appear in the borders of Workbench windows. However, there are several other types of standard gadgets used by Amiga programs. You will encounter these gadgets as you proceed through this manual and learn about the programs on the Workbench2.0 disk. This section briefly describes these gadgets and how to use them. When a gadget is used in a program window, the documentation explaining that program will also explain the exact function of the gadget.

NOTE: Many of the examples used in this section refer to editors in the Prefs drawer. Don’t worry if you do not understand the concept behind an editor; they are further explained in Chapter 3, “Preferences”. Right now you should only be concerned with learning how to use the different types of gadgets.

Action Gadgets

Action gadgets allow you to make a choice between two or more alternatives. Selecting an action gadget carries out your decision and closes the window. You have already seen action gadgets in the requesters that appear when you copy or format a disk. These gadgets are labeled Continue and Cancel.



Many programs use action gadgets to allow you to Save or Use changes, confirm that information is OK, or Continue with a procedure. You will also be presented with a Cancel or Quit gadget that lets you exit the operation safely.

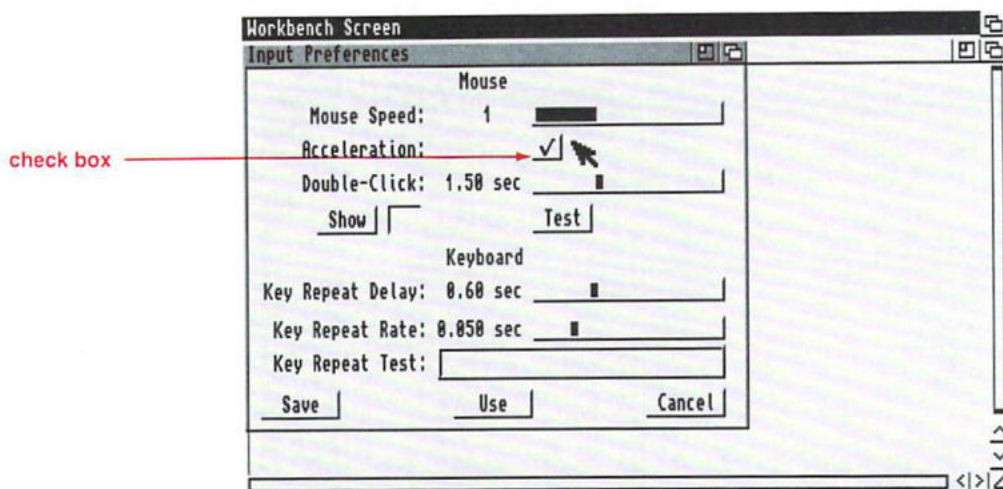
To select an action gadget:

1. *Point to the action gadget.*
2. *Click the selection button.*

Check Box



Check boxes let you turn an option on or off. For instance, the Input editor uses a check box to allow you to turn on the Acceleration option.



When the option is on, the box contains a check mark. If the option is off, the box is empty.

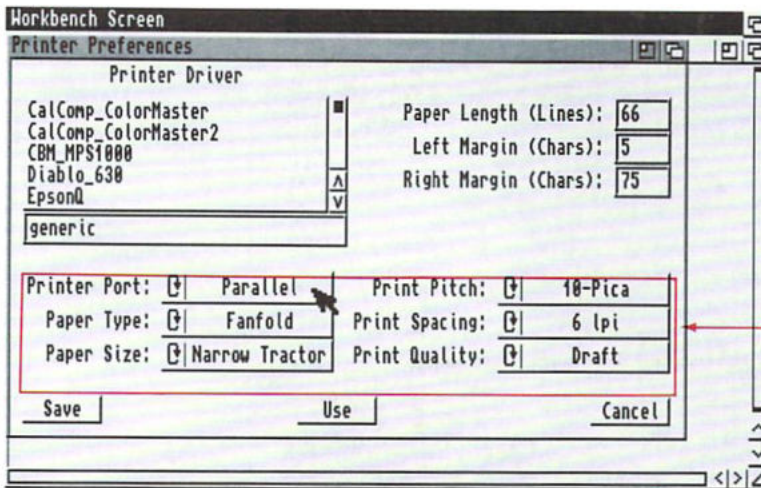
To change the setting:

1. *Point to the check box.*
2. *Click the selection button.*

Cycle Gadget



A **cycle gadget** lets you select one option from a list of options. The displayed option is the selected option. For instance, the Printe editor contains cycle gadgets that let you select your printer specifications.



To see the available options:

1. *Point to the cycle gadget and click the selection button.*

The next option in the list will be displayed.

2. *Keep clicking the selection button until you return to the first option that was displayed.*

When you return to the first option, you will know that you have read through all the available choices.

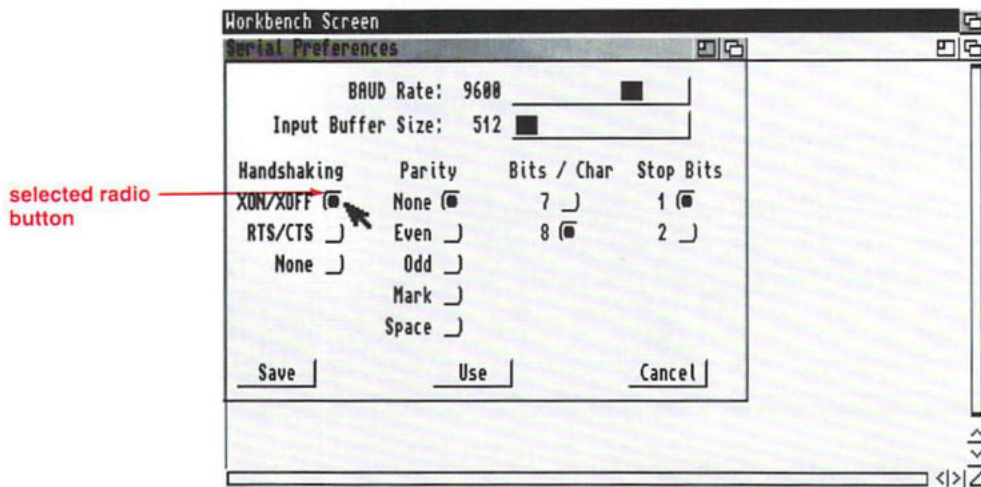
To select an option:

1. *Click on the cycle gadget until the option you want to select is displayed.*

Radio Button



A **radio button** also allows you to select one option from a list. However, in this case, the entire list is visible and each option has a radio button next to it. For instance, the Serial editor uses radio buttons to let you select the appropriate settings for sending information through the serial port (fully explained in Chapter 3).



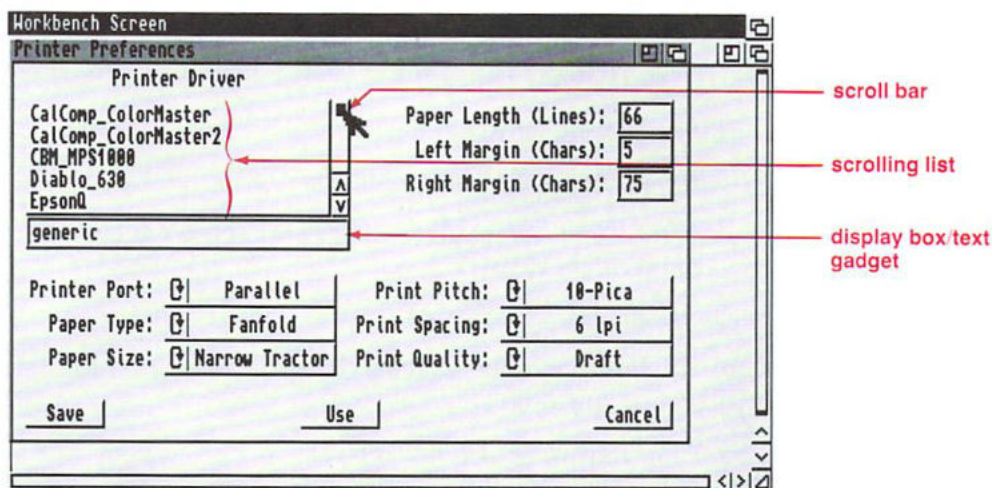
The radio button next to the selected option will be highlighted, and it will appear that the button has been pushed into the screen. The other buttons in the list will remain one color and will appear to be raised above the screen.

To select a radio button:

1. Point to the radio button next to the option of your choice.
2. Click the selection button.

Scroll Gadget

Scroll gadgets used within windows are very similar to the scroll gadgets contained in the border of the Workbench windows.



However, this type of scroll gadget allows you to select from options displayed in the **scrolling list**. It may also have a **display box** or text gadget underneath the scrolling list that shows the selected option.

The scroll gadget shown above is from the Printer editor. This gadget allows you to choose the type of printer you have attached to your Amiga.

A scroll gadget can only show a limited number of options at a time, but there may be many more from which you can choose. The available choices will be shown in the scrolling list. You can tell if all the options are shown by looking at the scroll bar. If all the options are visible, the scroll bar fills the entire scroll box. If the scroll bar only fills part of the scroll box, not all of the options are visible.

To scroll through the options:

1. *Point to the scroll bar.*
2. *Hold down the selection button.*
3. *Move the mouse so that you drag the scroll bar through the scroll box.*

To choose an option:

1. *Scroll through the list until the option you want to choose is displayed.*
2. *Point to your choice.*
3. *Click the selection button.*

Your choice will be highlighted. When you release the button it will appear in the display box or text gadget underneath the scroll area.

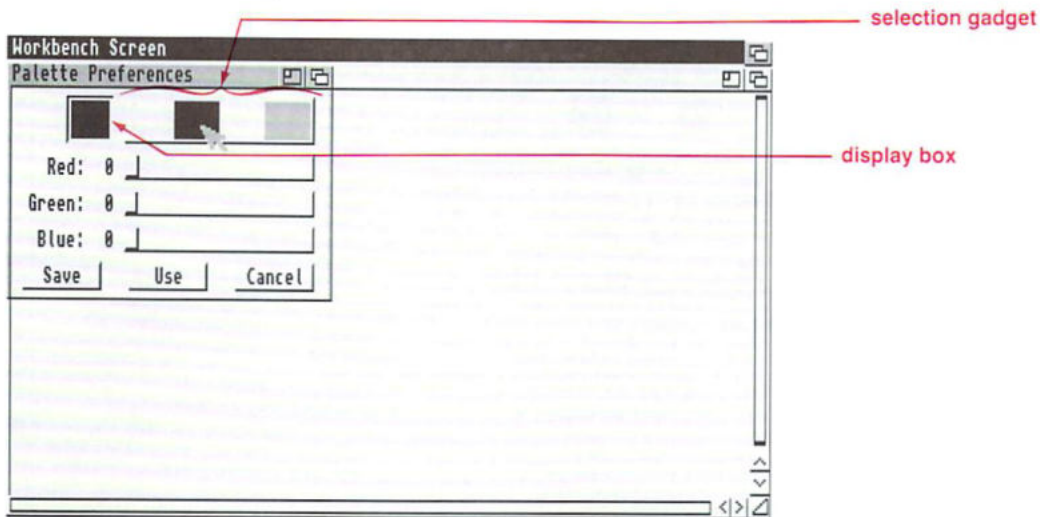
A display box is a rectangular box, similar in appearance to a text gadget, but you cannot enter information in it. A display box only reflects the choice you made with the scroll gadget.

When a text gadget is underneath the scroll gadget, you can sometimes enter a choice not displayed in the scrolling list, such as a new filename for saving information.

Whether or not the scroll gadget uses a display box or text gadget depends on the individual program.

Selection Gadget

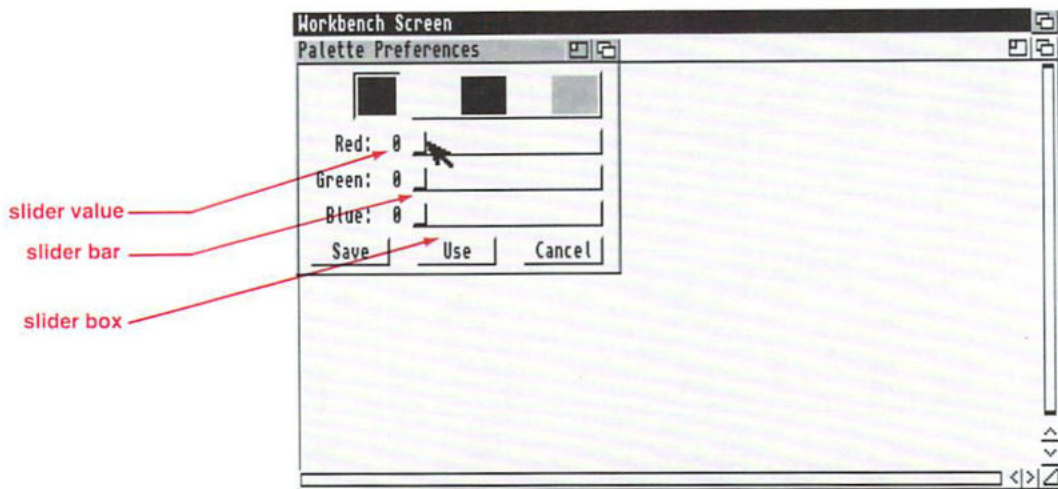
A **selection gadget** lets you select from several displayed options. The Palette editor uses a selection gadget to let you select a color to change.



In this case, you simply point to the color you want to use, and click the selection button. The selected option will appear in the display box to the left of the selection button.

Slider Gadget

Slider gadgets allow you to select a value within a given range. They are similar to scroll gadgets in that you drag a slider bar through a slider box to select a specific value.



The sliders shown above allow you to change the colors of the Workbench.

The **slider value** is shown to the left of the slider. This is the value associated with the current position of the slider bar.

To change the value:

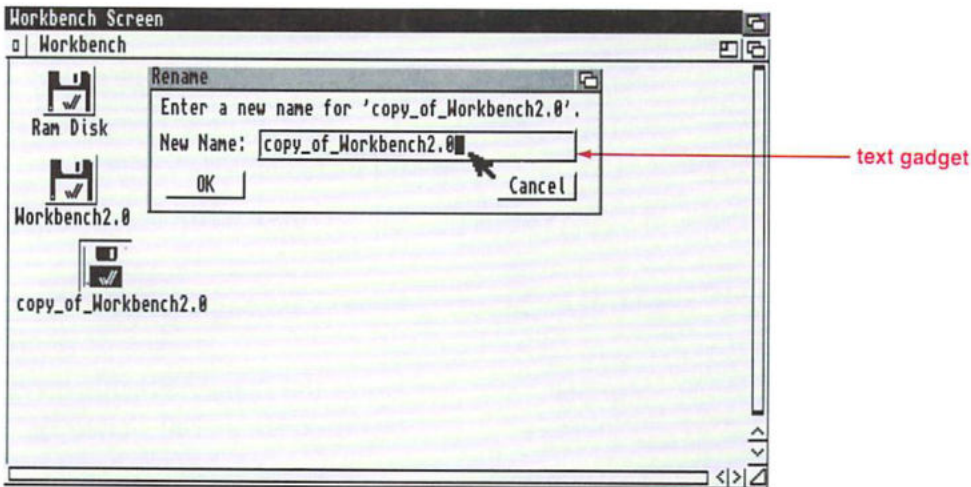
1. *Point to the slider bar.*
2. *Hold down the selection button and move the mouse to the right or left.*

The slider value will change as the bar moves through the box.

3. *When the desired value is shown, release the selection button.*

Text Gadget

A text gadget allows you to enter text that will be used by a program. You've already seen a text gadget in the requester that appears when you choose the Rename menu item.



To enter text in a text gadget:

1. *Select the gadget.*

Point to the gadget, and click the selection button.

2. *A cursor will appear in the gadget.*

A cursor is a small box that indicates where the next typed action will occur. When you type, the text appears to the left of the cursor.

3. If there is already text in the gadget, delete it using Backspace.

Some other keys you can use to edit the text in the gadget include:

Del	Erases the character highlighted by the cursor.
right Amiga-X	Erases all the text in the gadget.
right Amiga-Q	Retrieves what was in the gadget before the text was changed.
Shift-left cursor	Moves cursor to the beginning of the line.
Shift-right cursor	Moves cursor to the end of the line.
Shift-Del	Erases the character highlighted by the cursor and all characters to the right of the cursor.
Shift-Backspace	Erases all the characters to the left of the cursor.

4. To enter new text, simply type on the keyboard.

The characters you type will appear in the text gadget.

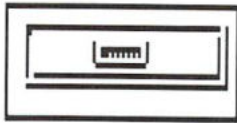
5. When the text gadget reflects the correct information, press Return.

Icons

Icons are pictures on the screen that represent disks, drawers, and files. Icons provide quick access to information stored on a disk. The Workbench uses several types of icons:



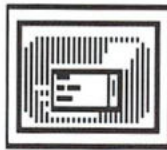
A **disk** icon represents any disk that is available or accessible by the Workbench, such as a floppy disk, hard disk, or Ram Disk. Disk icons are located in the Workbench window. When you open a disk icon, a window appears on the screen.



A **drawer** icon represents a subdivision of the disk storage area. When you open a drawer icon, a window appears.



A **tool** icon represents a specific program. For example, the Clock icon in the Utilities drawer is a tool. When you open a tool icon, the program is started.



A **project** icon represents a file where information created or used by a tool is stored. The Mode_Names icon in the WBStartup drawer is an example of a project. When you open a project icon, the associated tool, if any, is also opened. The tool will then begin to operate on the project.



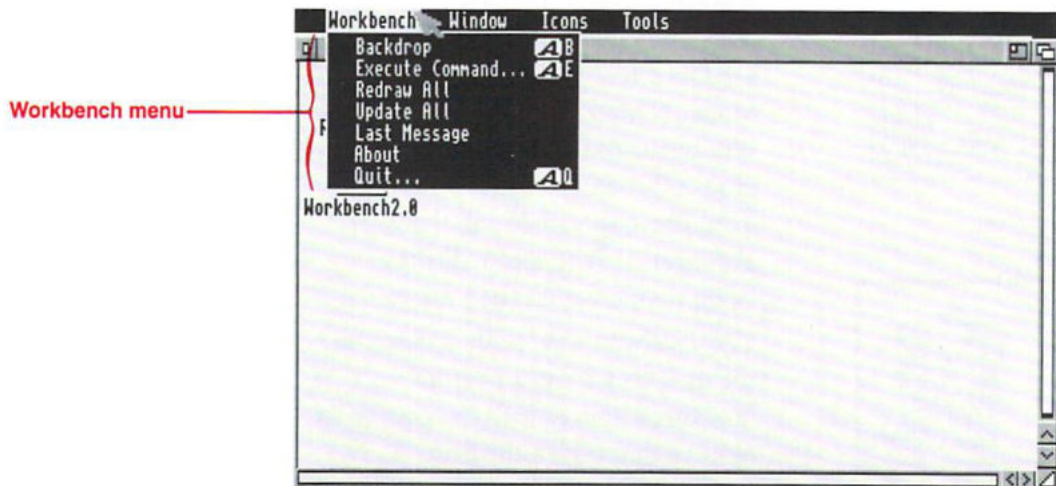
The **Trashcan** represents a place on the disk that is used to store unwanted items until you choose to remove them from the disk.

Every icon on your screen, whether it's a disk, drawer, project, tool, or trashcan, has a corresponding file that contains the information that produces the icon's image. These files are called **.info** files.

For instance, the Clock icon, in the Utilities drawer, represents the Clock program. There are two files in the Utilities drawer: Clock, which contains the data to run the program, and Clock.info, which contains the data that creates the Clock icon.

The Workbench Menu

The Workbench menu pertains to the Amiga's general operations as well as to all open windows on the Workbench screen. You can use the Workbench menu to update the screen display or see which version of the software you are using. You can even use it to execute AmigaDOS commands. (See Chapter 7, "Using AmigaDOS," for more information.)

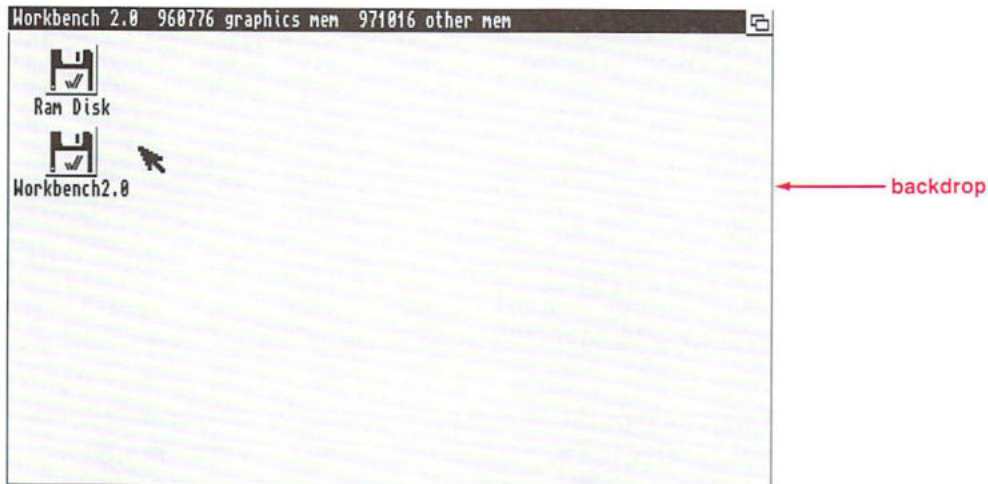


Many of the menu items have **keyboard shortcuts** which are shown to the right of the item. The shortcuts are an alternative to using the mouse. To choose the menu item with the keyboard, hold down right Amiga, then press the specified letter key. In this manual, the keyboard shortcuts are shown along the right margin.

Backdrop

AB

When you choose Backdrop, the Workbench window disappears and the disk icons appear on the Workbench screen. The disk icons are no longer in a window. This is useful when you have several windows open, and you need to move through them frequently. It eliminates the need to keep moving the Workbench window out of the way.

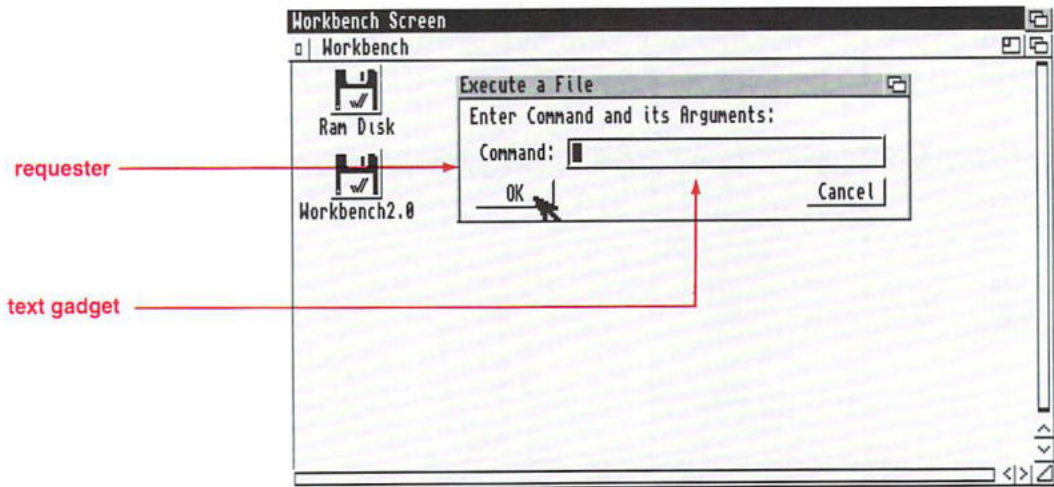


When Backdrop has been chosen, a check mark appears to the left of the menu item. To return to the Workbench window, choose the Backdrop menu item a second time.

If you choose Backdrop, then turn off or reboot your computer, the Workbench window will reappear. To save your Backdrop selection, use the Snapshot menu item in the Windows menu (explained on page 2-63).

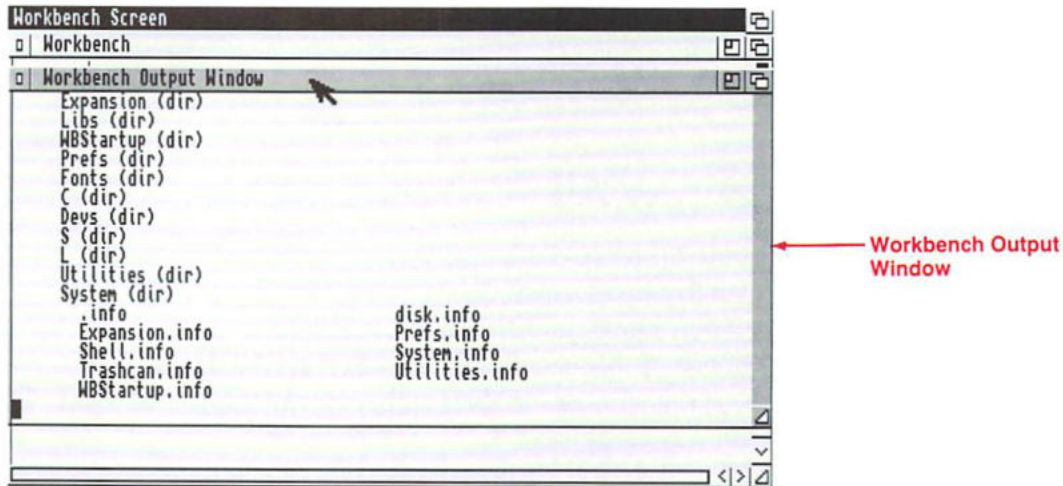
Execute Command... AE

This menu item allows you to **execute** (start) an AmigaDOS command without opening a Shell window. When you choose Execute Command, a requester prompts you to enter the command and its arguments. (The AmigaDOS commands are fully explained in Chapter 8, "AmigaDOS Reference.")



When you type the command, it appears in the requester's text gadget. After you've entered the command, select the OK gadget (or press Return) to execute it. If you select the Cancel gadget, the requester will disappear without executing the command.

In cases where the command results in output, the Workbench Output Window is automatically opened on the front of the screen.



The window will remain on the screen until you select its close gadget.

If you choose Execute Command a second time, the text gadget will display the previously entered command. You must delete the old command before entering a new command. The output will be shown in a new Workbench Output Window, even if another output window is already open on the screen.

Here's an example of its use:

1. Choose Execute Command from the Workbench menu.

A requester with a text gadget will appear on the screen.

2. Type:

DIR Workbench2.0:

If you have a hard disk, type DIR System2.0:.

DIR is an AmigaDOS command which lists the contents of a disk or drawer.

3. Press Return.

A list of the Workbench2.0 disk's contents will be generated in the Workbench Output Window.

4. Close the Workbench Output Window by selecting its close gadget.

Redraw All

Redraw All redraws all open windows on the Workbench screen in case of a disturbance to the Workbench. On rare occasions a program may cause part, or all, of the screen to be disrupted. If this occurs, choosing Redraw All may help to restore the windows to their proper appearance.

Update All

Update All redraws each open window, updating its appearance to reflect the current state of the window. If you are only communicating with the Amiga through the Workbench, you probably will not use this menu item too often. However, if you are using both the Shell and the Workbench, you will find this option quite helpful.

If you have several windows open and have been using the Shell to make changes to the contents of the disk, the changes will not be immediately reflected in the windows. For instance, if you were using the Shell to delete files and their icons, the icons would remain in the windows until you closed the windows and re-opened them, or chose Update All from the Workbench menu.

Last Message

Sometimes you will see a message flash across the title bar; this may be either an information or error message. Some examples of common error messages are: object not found (the file you are looking for is not on the disk), disks are incompatible types (appears if you try to drag a floppy disk icon over the Ram Disk icon or hard disk icon), the Trashcan cannot be moved (appears if you try to drag the Trashcan out of its window).

Some messages flash briefly, others remain until you press the selection button. In order to see the last message, select the Workbench window and choose Last Message from the Workbench menu. The message will be shown in the title bar. AmigaDOS error messages are explained in Chapter 8, "AmigaDOS Reference."

About

The About menu item opens a requester which shows the internal version number of the Workbench and Kickstart software as well as copyright information. Select the OK gadget to close the requester.



Quit...

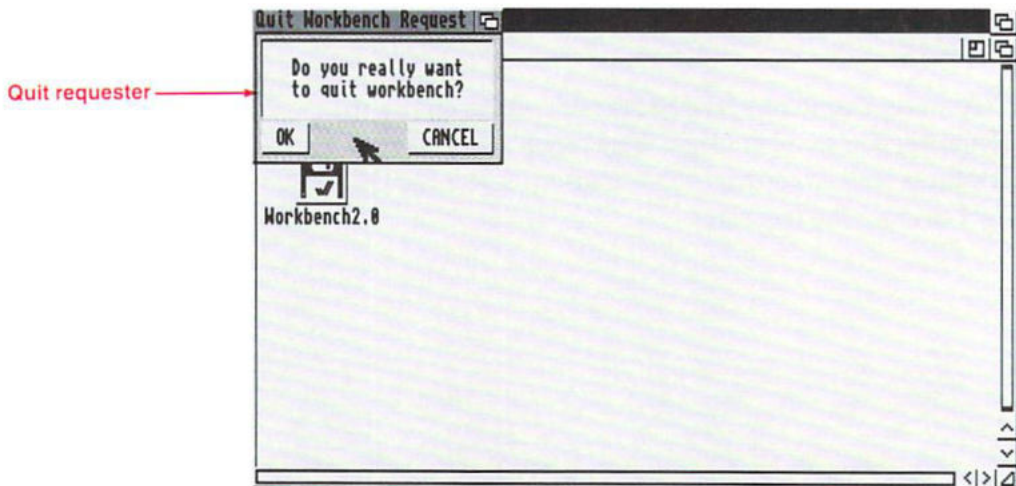
AQ

This operation is not recommended unless you are an experienced Shell user.

Quit allows you to close all Workbench operations. However, if you have any Workbench programs running when you choose the Quit menu item, an error message will flash in the title bar of the screen. The message will state that the system cannot quit as there are Workbench programs launched. It will also state the number of launched programs.

If you still want to Quit, you must shut down all programs. Disk and drawer windows can remain open.

Once all programs are terminated, you can choose the Quit menu item. This time, a requester will ask if you want to quit the Workbench.



If you select OK, all Workbench windows and icons will disappear, and you will not be able to access any of the Workbench menus. The only way you will be able to communicate with the Amiga is through a Shell window that was opened with Execute Command or through another Shell. A Shell window opened from the Shell icon is considered a Workbench program, and you cannot use Quit if the window is open.

To Quit the Workbench and leave a Shell window open, use the Execute Command menu item:

- 1. Choose the Execute Command menu item.***

A requester will appear on the screen.

- 2. Type NEWSHELL in the requester, and press Return.***

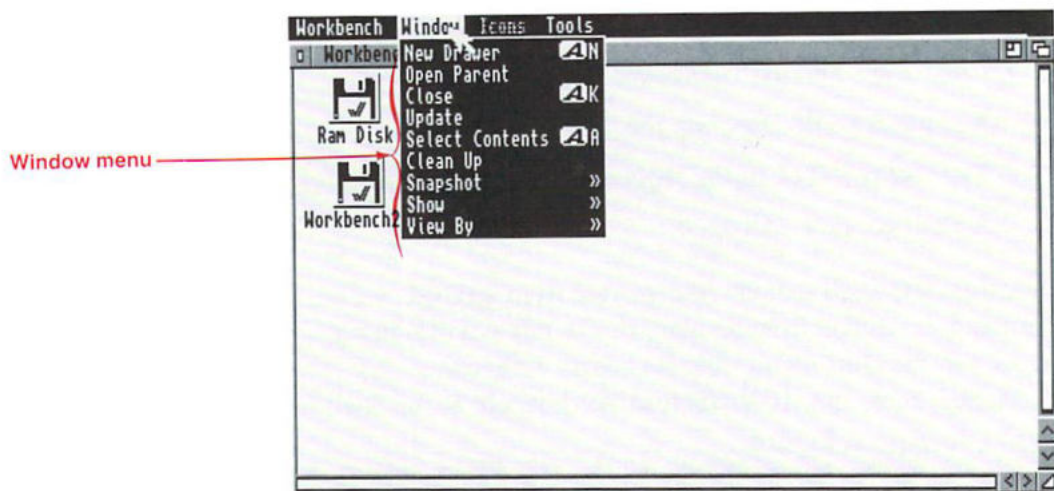
A Shell window will open.

Since this new Shell window was started from a typed command, instead of from an icon, this window will remain open when the Quit menu item is chosen. To get the Workbench back, type LOADWB (load Workbench) at the Shell prompt, then press Return.

If you do not leave a Shell window open, you have to reboot the Amiga to return to the Workbench system.

The Window Menu

The Window menu is only available when a window on the screen is selected. The options in the Window menu pertain to the currently selected window. For instance, you can use the window menu to organize the contents of a window or to change the way the information is displayed.



Some of the menu items have keyboard shortcuts which are shown to the right of the item. To choose a menu item with the shortcut, press right Amiga and the specified letter key.

New Drawer

AN

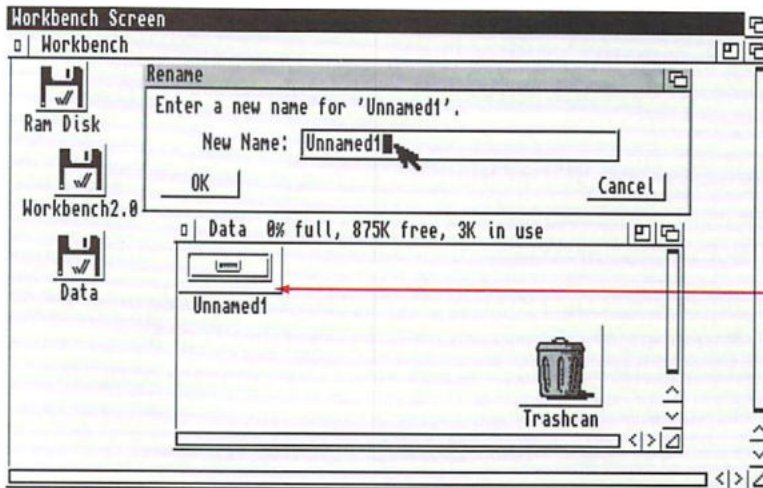
New Drawer allows you to create a new drawer in a window. The drawer, which is labeled Unnamed1, is created in the currently selected window. This is a convenient way to create drawers for file storage.

You cannot create a new drawer in the Workbench window.

To create a New Drawer:

1. Select the window in which you want to create the drawer.
2. Choose New Drawer from the Window menu.

A new drawer icon, labeled Unnamed1, will appear in the window.



3. A Rename requester also appears to allow you to change the name of the drawer.

Delete the existing name, type in the new name, and press Return.

Open Parent

Except for the Workbench window, every window has a parent window. The parent window is the window that contains the icon that was opened to create the current window.

For instance, the Workbench window is the parent of all the disk windows. The Workbench window contains the disk icons that must be opened in order for the disk windows to appear.

Disk windows often contain drawers. When you open a drawer icon, a window appears. Therefore, the disk window is the parent of the drawer window.

For example, the Workbench2.0 disk window contains the Utilities drawer. When you double-click on the Utilities drawer icon, the Utilities window is opened. The Workbench2.0 disk window is the parent of the Utilities window.

Choosing Open Parent brings the selected window's parent to the front of the display. If the parent window is closed, it is automatically opened.

For instance, if the Utilities window is selected and you choose Open Parent, the Workbench2.0 disk window will be automatically opened (if it was closed) and brought to the front of the display. If the Workbench2.0 disk window is selected and you choose Open Parent, the Workbench window will be brought to the front of the display.

To open the parent of a window:

- 1. Select the window.*
- 2. Choose Open Parent from the Window menu.*

Close

AK

To remove a window from the screen, choose Close.

1. *Select the window.*
2. *Choose Close from the Window menu.*

The window will disappear.

Mouse Shortcut: A shortcut for closing windows is to select the close gadget in the upper left corner of the window.

Update

If you make changes to a window through the Shell or the Execute Command menu item, those changes will not be reflected in the window until you either close and re-open the window or choose the Update menu item. Update redraws the selected window so that it accurately reflects the contents of the window.

(This is very similar to the Update All menu item in the Workbench menu, except that Update only affects the currently selected window.)

In the following example you will delete a drawer from the Workbench window using the Execute Command menu item. Then, you will use Update to correct the contents of the window. However, you will need a drawer to delete, so create an extra drawer with the New Drawer menu item.

1. *Open the Ram Disk window.*
2. *Choose New Drawer from the Window menu.*

A new drawer named Unnamed1 is created in the Ram Disk window. You have created a new drawer,

Unnamed1, and a new file, Unnamed1.info, on your Ram Disk. The Unnamed1.info file contains the data needed to create, or draw, the Unnamed1 drawer icon.

3. *When the Rename requester appears, just press Return.*

4. *Choose Execute Command from the Workbench menu.*

When the requester appears, type:

Delete RAM:Unnamed1.info

Then press Return. The Workbench Output Window will display a message stating that the Unnamed1.info file has been deleted. However, the icon will still remain in the Ram Disk window.

5. *Select Update from the Window menu.*

This redraws the window so that its contents accurately reflect the current state of the disk. The Unnamed1 icon will be removed from the Ram Disk window.

However, you will still have a drawer called Unnamed1 on the Ram Disk. To delete it, choose Execute Command again, type Delete RAM:Unnamed1 in the requester, then press Return.

Select Contents

AA

When you choose Select Contents, all of the icons in the active window are selected. This is an alternative to drag selection or extended selection.

Clean Up

Clean Up rearranges the icons in a window in an orderly fashion. When icons are copied or created, they sometimes appear on top of another icon or in a separate area of the window. Clean Up automatically places all the icons in the selected window in a neat arrangement, so that you do not have to arrange each icon individually.

Clean Up does not save the arrangement to disk. If you only clean up a window, but don't save it with the Snapshot menu item (explained below), the arrangement will be lost the next time you open the window.

To Clean Up a window:

- 1. Select the window you want to rearrange.*
- 2. Choose Clean Up from the Window menu.*

Snapshot



Snapshot lets you save the arrangement and position of a window. It is commonly used after Clean Up. When you point to the Snapshot menu item, two submenu items appear: Window and All.

Snapshot Window lets you save the position and size of the selected window as well as the Show and View By modes (explained in the following sections). However, it does not save the position of the icons in the window.

To save the placement of a window:

- 1. Select the window.*
- 2. Choose Snapshot Window from the Window menu.*

Snapshot All lets you save the positions of all the icons as well as the position and size of the selected window. Whenever you open the window, it will be arranged in the same way.

To save the placement and the arrangement of a window:

1. *Select the window.*
2. *Choose Snapshot All from the Window menu.*

Show



Not every file on a disk or in a directory has a corresponding icon. The Show menu item allows you to see all the files on a disk, whether or not they have icons. Show has two submenu items: Only Icons and All Files.

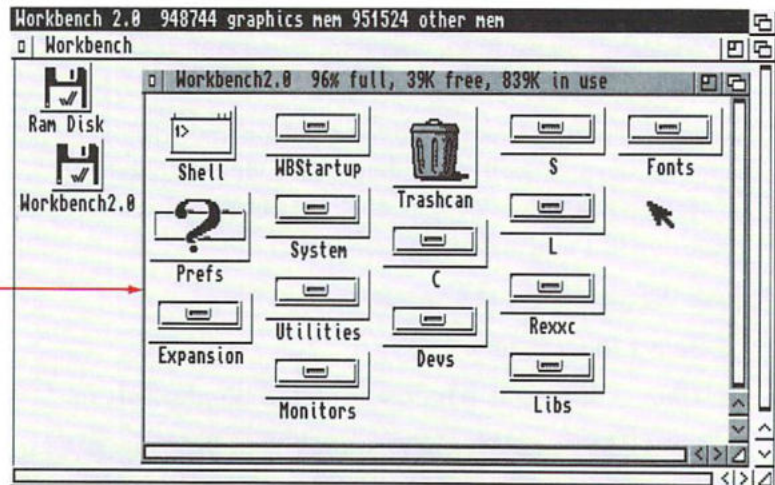
Pseudo-icons do not have .info files.

Choosing Show All Files displays a **pseudo-icon** for each file or drawer in the selected window. (A pseudo-icon is a temporary icon supplied by the Workbench for files that do not have their own icons.) You can treat these pseudo-icons just like any other icon and use the menu items in the Icons menu to manipulate the icon.

To display icons for all the files in a window:

1. *Select the window.*
2. *Choose Show All Files from the Window menu.*

after choosing
Show All Files

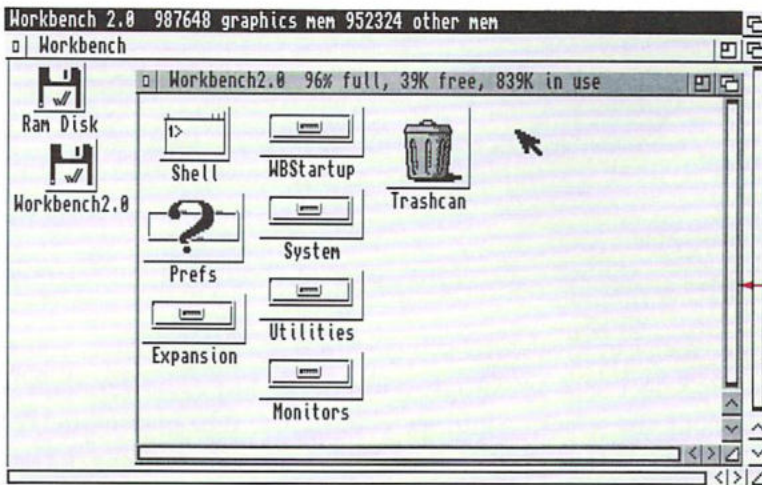


Show Only Icons displays only those files and drawers which have icons (.info files). All pseudo-icons will be removed from the window.

To display only the real icons:

1. *Select the window.*
2. *Choose Show Only Icons from the Window menu.*

Only the window's real icons will be displayed.



after choosing
Show Only Icons

View By



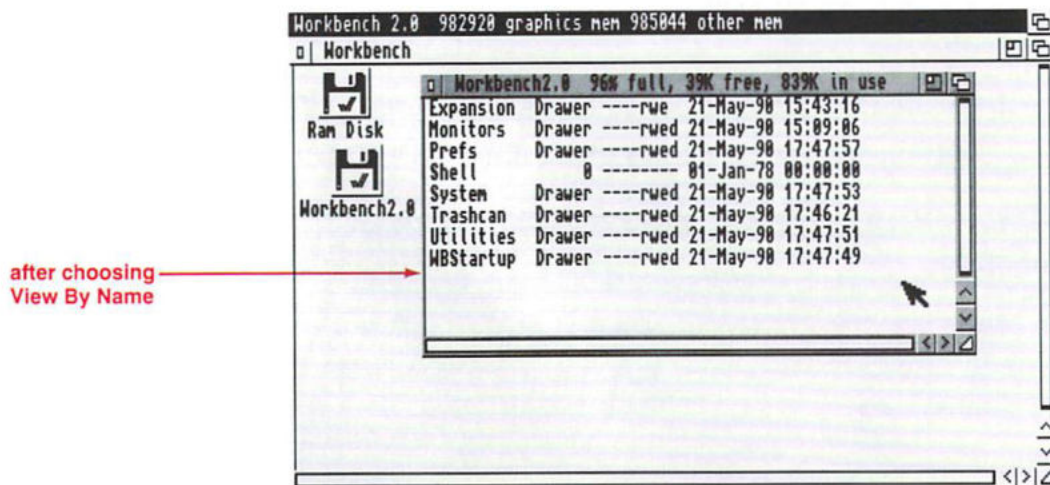
The View By menu item allows you to change how the information in a window is displayed. View By has four submenu items: Icons, Name, Date, and Size.

When you choose View By Icons, the window appears in its **default** state.

Choosing View By Name changes the window display. The window will contain an alphabetical list of the icons. This list includes the size of the file, its **attributes** (whether it can be

Default is a term used to describe the standard setting decided by the system if the user does not specify an alternative.

read, deleted, executed, or written), and the date it was created. (See the explanation of the Information menu item, page 2-76, for details on attributes.)



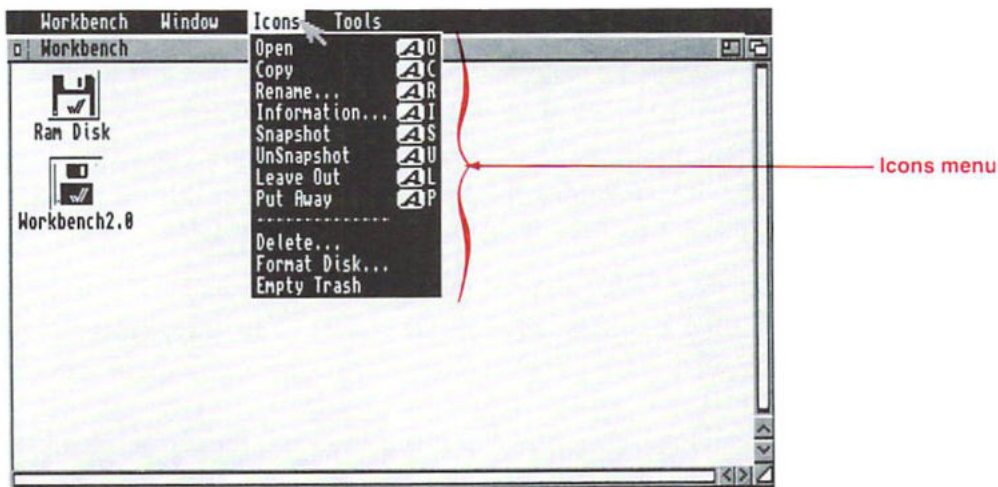
You can select files and drawers from the window just as you would select an icon. Simply point to the file or drawer name, and click the selection button. The name will be highlighted to show that it is selected. You could then use menu items in the Icons menu to manipulate the file or drawer. To open a file or drawer, point to the name, and double-click the selection button.

View By Date sorts the list in chronological order, with the most recently created file listed first. This is helpful if you have two different versions of a file and are looking for the one with the most current information.

View By Size sorts the list by size, with the smallest file listed first. If you find that you are running out of room on your disk, you can use this option to see which files take up the most space. You can then choose which files to delete or to move to another disk.

The Icons Menu

The Icons menu lets you work with the icons on the screen. Among other things, you can copy, rename, and open icons with this menu. You can also delete icons, and the corresponding files or drawers, from the disk. An icon must be selected before you can choose items from this menu.



Open

AO

To **open** an icon is to make the items represented by the icon available for use. When you open a disk or drawer icon, a window appears on the screen displaying the icons contained on that disk or in that drawer. When you open an individual project or tool, you actually start the corresponding program.

To open an icon:

1. *Select the icon.*
2. *Choose Open from the Icons menu.*

Mouse Shortcut: A shortcut for opening icons is to point to the icon and double-click the selection button. This is faster than using the menu.

Copy

AC

Copy allows you to copy disks, drawers, programs, or files. A copy of a drawer, tool, or project is made in the same window as the original. (To copy a drawer, tool, or project to another disk, see the "Copying by Dragging" section, page 2-73.)

An important use of Copy is for making backup copies of your disks. If your Amiga only has one disk drive, this is the most common way to copy disks using the Workbench.

To copy a disk:

The disk that is being copied is known as the source disk (FROM disk). You should always write-protect your source disk so that you cannot accidentally erase any of its contents. (The write-protect tab should be pushed towards the top of the disk so that the small hole is uncovered.)

The disk that you are copying to is known as the destination disk (TO disk). This can be a blank disk or a previously used disk whose contents you no longer need. This disk must be write-enabled in order to accept the information from the source disk. (The write-protect tab should be covering the small hole in the corner of the disk.)

When you copy a disk using the Copy menu item, the Amiga only uses one disk drive even if you have a second drive connected to your Amiga.

When you use the Copy menu item, the system will read a certain amount of information from the source disk into the Amiga's internal memory. Then you will have to take the source disk out of the disk drive and insert the destination disk. (This is known as swapping disks.) The Amiga will then copy the information to the destination disk.

1. Put the disk you want to copy, the source disk, into the Amiga's internal disk drive, known as DF0:

Make sure the write-protect tab is in the protected position. The small hole in the corner of the disk should be open.

2. Select the source disk's icon.

Point to the disk icon, and click the selection button.

3. Choose Copy from the Icons menu.

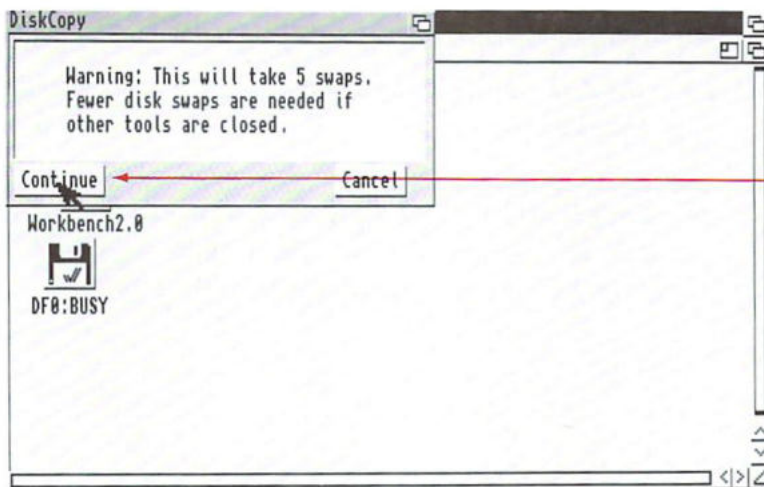
A requester will ask you to insert the Workbench2.0 disk in any drive. The system needs to read a program from the Workbench2.0 disk before it can begin the copy procedure.

If you have a hard disk, you will not see this requester since the system can read the program off the hard disk. Skip ahead to step 5.



4. Insert the Workbench disk.

If the disk copy is going to require several swaps (5 or more), a requester will tell you how many times you will have to swap the disks.



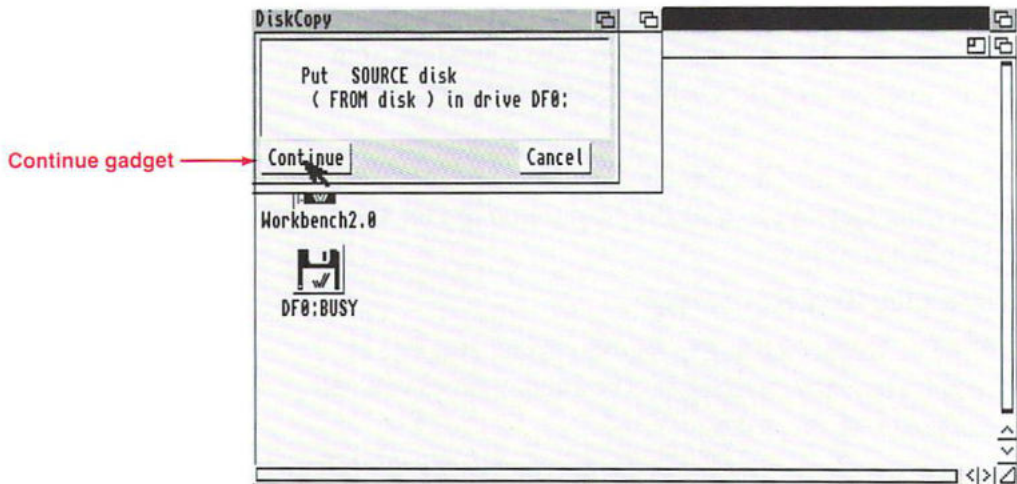
Continue gadget

Closing any unnecessary windows or stopping any unneeded processes will help to reduce the number of swaps.

If the disk copy requires fewer than 5 swaps, you will not see a requester.

5. Select the Continue gadget in the swap requester.

A requester tells you to insert the source disk into drive DF0:, the internal drive.

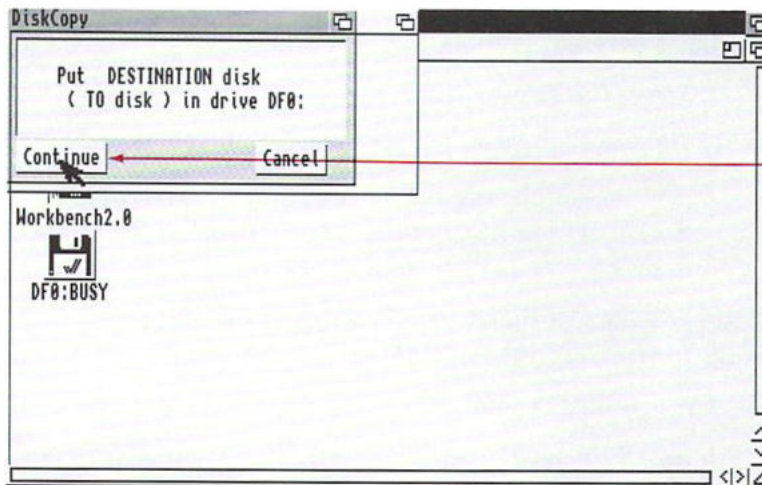


6. Put the disk you want to copy in the drive, then select the Continue gadget.

Text in the requester will reflect how many cylinders the system has read and how many are left to read. (A cylinder is a division of a disk. The 3.5 inch disks used with the Amiga have 80 cylinders numbered 0-79.)

A requester will instruct you when to insert the destination disk into drive DF0:.

Make sure the disk drive light is out before removing the source disk from the drive.



Continue gadget

7. Put your destination disk into the drive, and select the Continue gadget.

The data from the source disk will be copied to the destination disk.

You will have to follow the requesters and switch back and forth between the source disk and the destination disk as many times as stated in the first requester. (Be sure the drive light is out before ejecting a disk from the disk drive.) When the copy is finally finished, the message Disk Copy Finished appears in the requester.

If at some point you want to stop the copying process, wait for a swap requester to appear and select the Cancel gadget.

*You cannot
copy the
Trashcan.*

8. *Remove the destination disk from the drive and put a label on it.*

The destination disk's icon will now have the source disk's name with a copy__of__ prefix. For instance, if the source disk was called DataDisk the destination disk will be called copy__of__DataDisk.

To copy a drawer:

- 1. *Select the drawer icon.***
- 2. *Choose Copy from the Icons menu.***

The copy of the drawer, and any icons contained in the drawer, will be made in the same window as the original drawer.

To make a copy of the drawer on another disk, see the "Copying by Dragging" section.

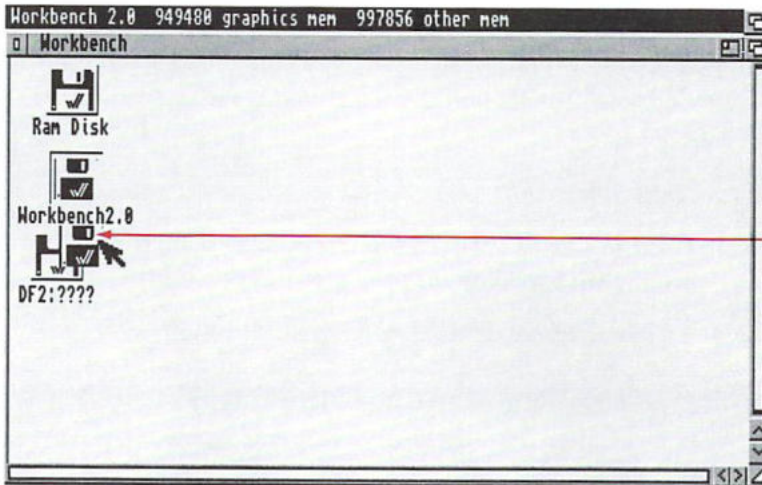
To copy a project or tool:

- 1. *Select the icon.***
- 2. *Choose Copy from the Icons menu.***

A copy of the icon will be made in the window.

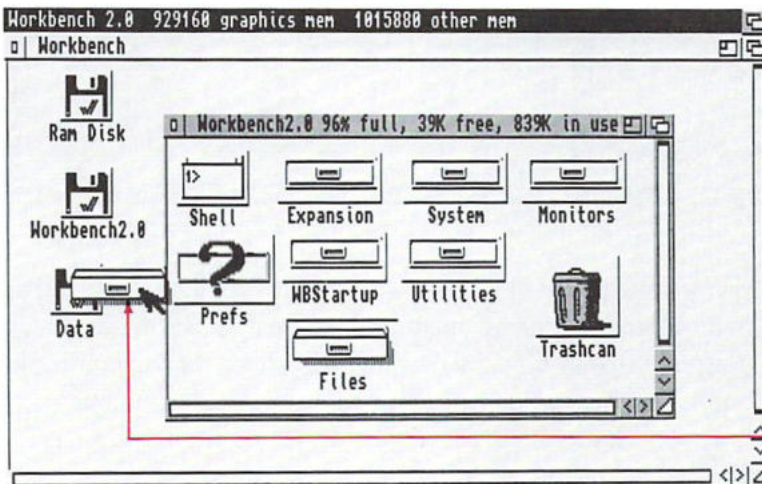
Copying by Dragging

You can copy a disk by dragging the source disk's icon over the destination disk's icon.



copying
Workbench2.0
disk

You can copy a drawer, project or tool to another disk by dragging the icon over the other disk icon or into the other disk's window. The original icon will stay on the original disk, and a copy will be created in the destination disk's window.



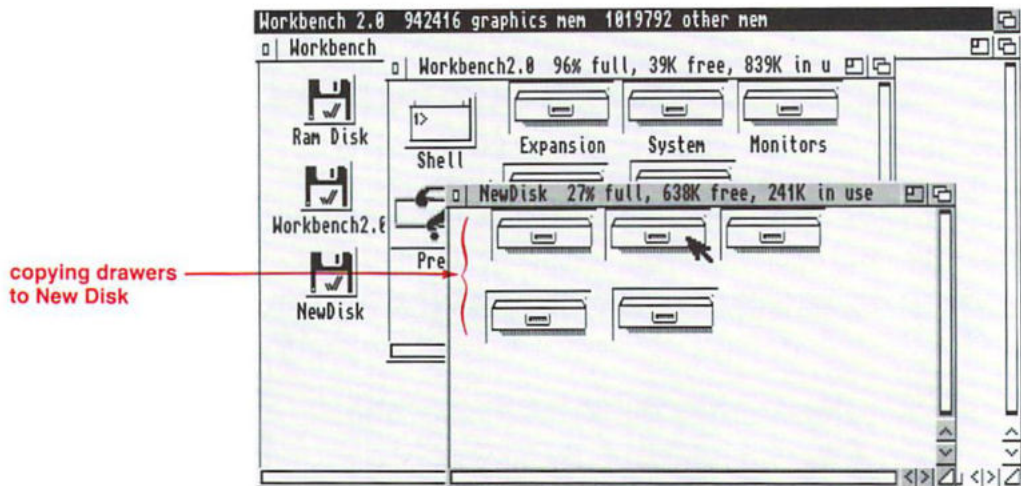
copying Files drawer
to Data disk

You cannot make a copy of an icon on the same disk with this method. For instance, if you were to drag an icon from the Utilities window into the System window, it would not be copied. The icon would simply change drawers. It would move from the Utilities drawer to the System drawer.

You can copy several icons at once by using drag selection or extended selection. When the icons you want to copy are selected:

1. *Hold down Shift.*
2. *Point to one of the selected icons, then drag it over the other disk's icon or into the other disk's window.*

As you drag one icon, the rest will follow.



If you have a hard disk, you can copy floppy disks into your hard disk partitions by opening the hard disk window (System2.0 or Work), and dragging the floppy disk icon into the window. When the copy is finished, a drawer for the floppy disk will be in your hard disk window. For more information, see Chapter 6, "Using a Hard Disk."

Rename...

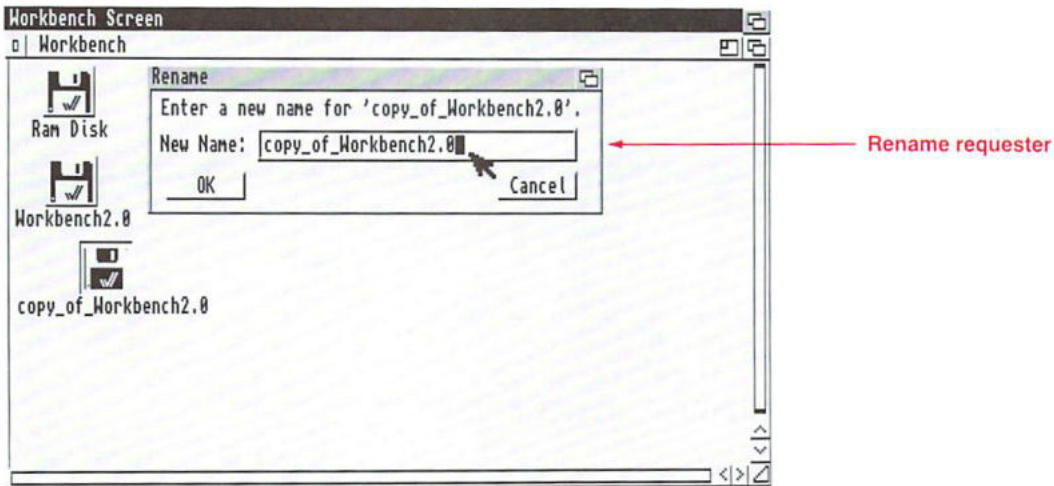
AR

You can change the name of an icon with the Rename menu item. A common reason for using Rename is to remove the `copy_of_` prefix from a newly created icon. You may also want to change the names of your disks and files as you create more files. For instance, if you originally created a disk called Reports and it is starting to get full, you may want to change its name to Reports1990. You could then start a new disk named Reports1991.

To rename an icon:

1. *Select the icon.*
2. *Choose Rename from the Icons menu.*

A requester with a text gadget will appear and show the current name of the icon.



3. Enter the correct name.

You will have to delete the old name (use Backspace) and enter the new name. Be sure to delete any spaces before or after the new name.

You can also press right Amiga-X to erase everything in the text gadget.

4. Press Return.

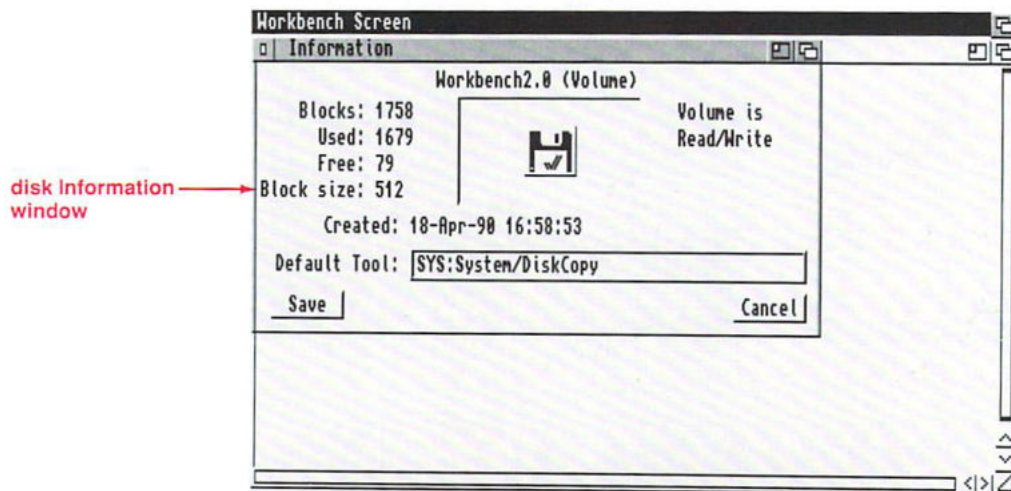
The requester will close, and the new name will appear under the icon.

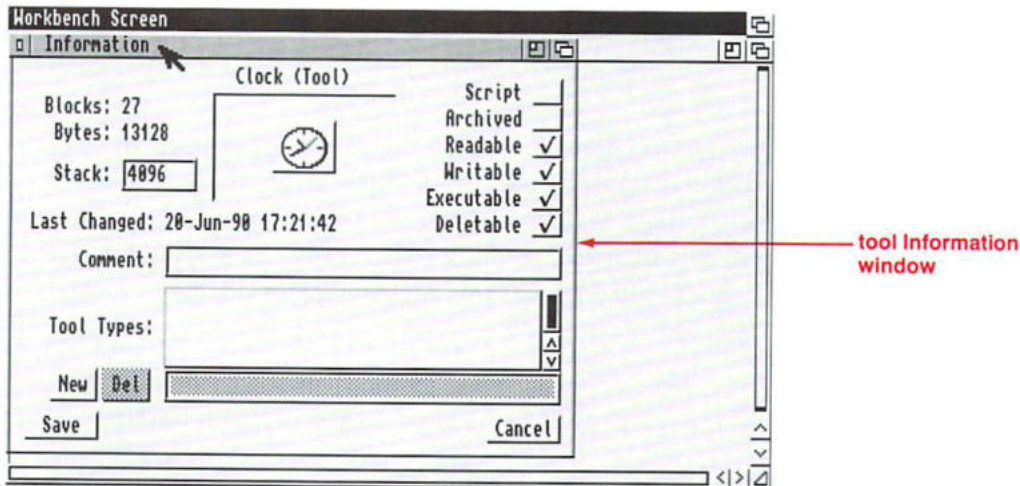
When renaming icons be careful not to leave any spaces before or after the new name. These will be impossible to discern on the screen and will cause confusion if you ever need to type the icon's name.

Information...

AI

Choosing Information results in a window displaying information about the selected icon.





This information includes the icon's:

- | | |
|-------------------|--|
| name | This is followed by the icon's type in parentheses. The permissible types of icon are disk (or volume), drawer, project, trashcan or tool. |
| image | A picture of the icon is displayed in the center of the window. |
| size | This is only given for a disk, project, or tool and is shown in blocks and bytes . |
| stack | This is the amount of memory reserved as temporary storage for a specific tool. (This does not apply to disk or drawer icons.) |
| last changed date | This indicates when the item represented by the icon was created or the last time it was changed. |
| status | For a disk, the status is either Read/Write (you can both read information from the disk and save |

If your Amiga does not have a battery backed-up clock, you must first set the correct time with the Time Prefs editor, explained in Chapter 3.

new information to it) or **Read Only** (you can only read information from the disk).

For a drawer, project, or tool, there are six independent attributes that you can select. When an attribute is selected, there will be a check mark in the box to its right. To select, or deselect, an attribute, point to its check box and click the selection button. Each attribute is explained below:

Script	If this attribute is selected and the program is executed through the Shell, it will be run as a script (a text file of AmigaDOS commands).
Archived	This attribute is set by some backup programs to let you know that a file or directory has been saved, or archived .
Readable	If this attribute is selected, you can read , or access, the information in a file.
Writable	If this attribute is selected, you can write information to the file. If it is not selected, you cannot make changes to the file. For instance, if a file is readable but not writable, you will be able to read its contents, but you will not be able to change them.
Executable	If this attribute is selected, you can execute, or run, the project or tool.
Deletable	If this attribute is selected, you can erase the drawer, project, or tool from the disk. If it is not selected, the object is protected from deletion.

If the icon is a project, there may be a **Default Tool**. This specifies the path to the tool that created the project. When the project icon is opened, the default tool is also opened so that it can work on the project.

If there is a Comments box, you can include a short note, up to 79 characters, in the information window. For instance, if you have an icon representing a text file you created, you may want to add a note to remind yourself of the file's contents. To do this, select the text gadget next to Comments, and type the note. Press Return when you are finished.

The **Tool Types** box allows you to specify different parameters for some programs or files. How to use Tool Types is covered in Chapter 4, "The Workbench Programs". If a program supports Tool Types, the permissible Tool Types will be explained when that program is explained.

To save any changes you make to the Information window, you must select the Save gadget in the lower left corner. If you make changes but decide not to save them, select the Cancel gadget or the window's close gadget.

Snapshot

AS

Snapshot saves the positions of all the currently selected icons on disk. Every time you open the window, any icons that you snapshot will appear in their saved positions. You can save the position of multiple icons by using drag selection or extended selection.

To snapshot the position of an icon:

1. *Select the icon(s) you want to snapshot.*
2. *Choose Snapshot from the Icons menu.*

The next time you open the window, the icon(s) will be in this position.

Unsnapshot

AU

Unsnapshot allows you to cancel the snapshot position of an icon.

To unsnapshot the position of an icon:

1. *Select the icon.*
2. *Choose Unsnapshot from the Icons menu.*

The next time you open the window, Workbench is free to place the icon anywhere it wants to within the window.

Leave Out

AL

You cannot use this menu item with the Trashcan.

Leave Out allows you to move an icon out of its original window and into the Workbench window. (The file represented by the icon remains in its original drawer on the disk, only the icon is moved.) The icon remains in the Workbench window, even if you reboot the machine.

For instance, you may have a file that you use every day, but you have to open a disk icon and two other drawers to get to it. For faster access, you can use Leave Out to move the icon into the Workbench window.

To use Leave Out:

1. *Select the icon.*
2. *Choose Leave Out from the Icons menu.*

The icon will move into the Workbench window.

Put Away

AP

After using Leave Out, you can return the icon to its original drawer by choosing Put Away.

To use Put Away:

1. *Select the icon in the Workbench window.*
2. *Choose Put Away from the Icons menu.*

The icon will move back into its original window.

Delete...

The Delete menu item lets you erase files, and their icons, from the disk. Use Delete with caution. Once you delete an icon, you cannot retrieve its information.

You cannot delete a disk icon or the Trash-can icon.

To delete an icon:

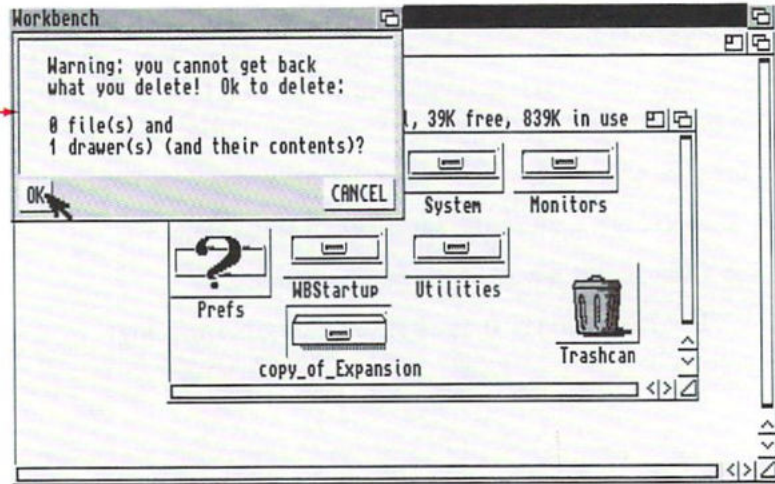
1. *Select the icon.*

You can use drag selection or extended selection to choose more than one icon to be deleted.

2. *Choose Delete from the Icons menu.*

A requester warns you that you cannot get back what you delete and shows the number of items to be deleted. This is to safeguard against deleting items that may still be selected from a previous operation.

deleting the
Copy_of_Expansion drawer



3. Select the OK gadget to delete the icon.

The icon will leave the screen and all data associated with that icon will be erased from the disk. If you do not want to delete the icon, select the Cancel gadget.

Be careful when deleting drawer icons. The requester will show the number of drawers being deleted, but you are also erasing everything contained in those drawers.

Format Disk...

In some application software manuals, you may see the word Initialize used as a synonym for Format.

When you insert a blank 3.5 inch disk into the Amiga's disk drive, the disk icon is labeled DF0:???, DF1:???, or DF2:???, depending on which disk drive it is in. At this point, the Amiga does not recognize the disk as an AmigaDOS disk. In order to use the disk, it must first be formatted for AmigaDOS.

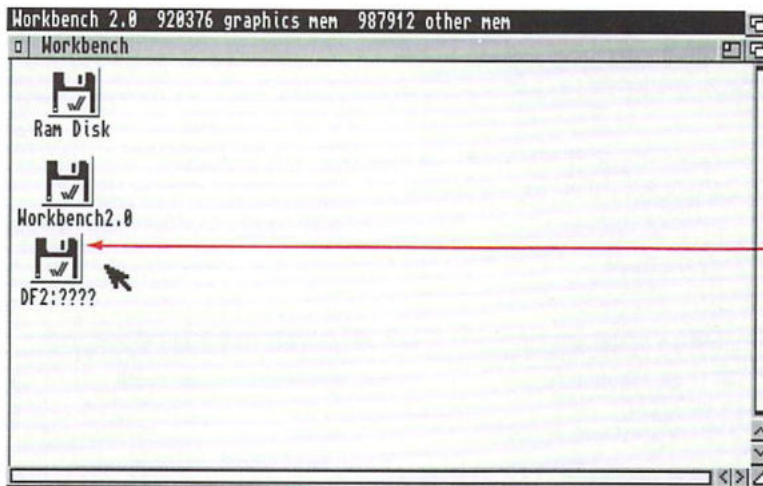
To format a disk:

1. Put the disk into the disk drive.

You can put the disk into any disk drive — internal or external. Make sure the disk is write-enabled (the plastic tab should be covering the hole).

2. Select its disk icon.

If it is a blank disk, the icon will be labeled according to which disk drive it is in (DF0:????, DF1:????, etc.).



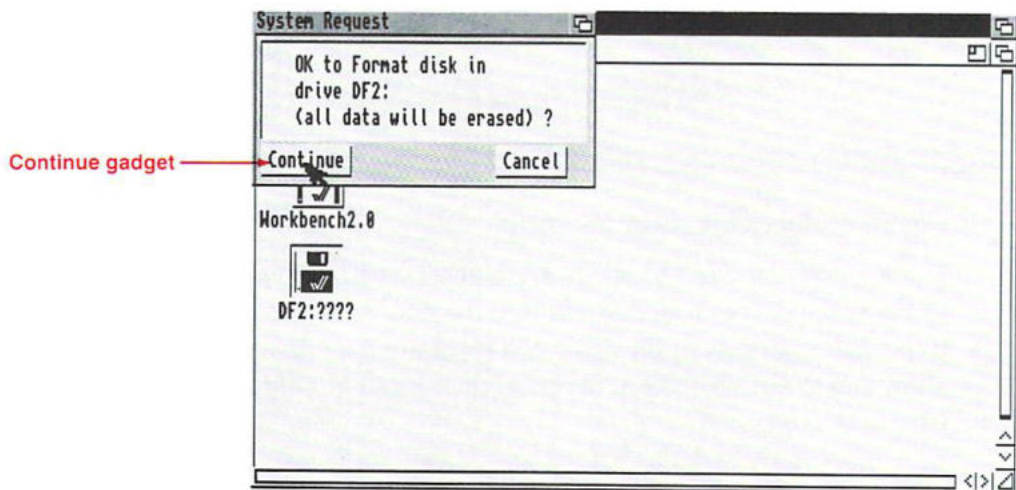
blank disk icon

3. Choose Format from the Icons menu.

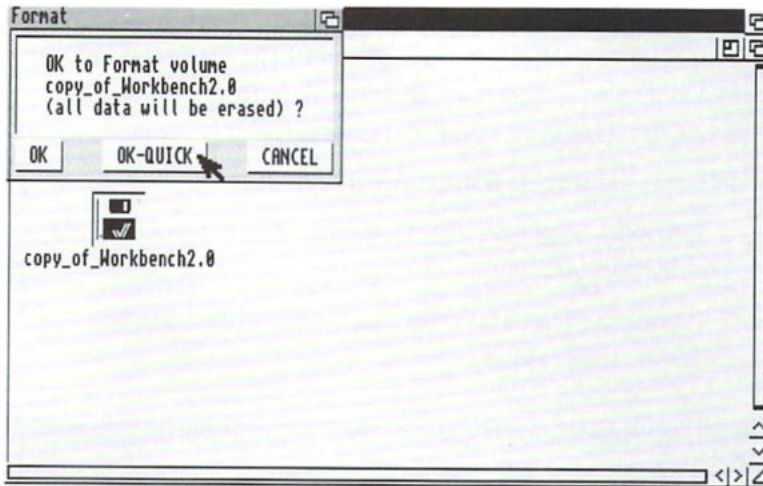
If you are formatting a blank disk, a requester will ask you to insert the disk to be formatted into the disk drive. Even if the disk is already in the drive, the requester appears to allow you to switch disks or to cancel the operation. Select Continue to proceed.



A second requester will ask if it is OK to format the disk in the disk drive and remind you that all the data will be erased. Select Continue to proceed or Cancel to abort the procedure.



If you are formatting a disk that contains data, a requester will ask if it is OK to format the disk, it will state the disk's name, and remind you that all the data will be erased.



This requester presents you with a third option: OK-Quick. If you select OK-Quick, just the **root block** track of the disk is formatted. The root block contains the information that identifies the disk and where all the files are stored. Erasing the root block essentially erases the entire disk as the system will no longer be able to find any of the files on the disk. This is much quicker than a regular format.

However, if your disk has any type of read/write error, you should perform a regular format by selecting the OK gadget. Each cylinder of the disk will be reformatted.

Once the formatting process begins, text in the requester shows the cylinder of the disk that is being formatted and verified. After a disk has been formatted, its disk icon will be labeled Empty. You can change the name by using the Rename item in the Icons menu.

Empty Trash

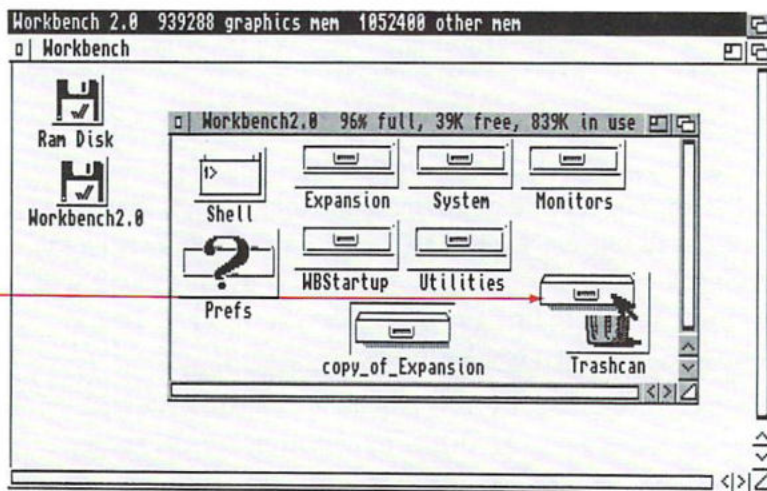
As explained in the "Icons" section, the Trashcan is a place where you store unneeded files. To delete an icon using the Trashcan, drag the icon you want to "throw away" over the Trashcan icon. The icon is then stored in the Trashcan drawer until you decide to delete it. To actually remove the file(s) stored in the Trashcan, you must choose Empty Trash.

To delete an icon with Empty Trash:

1. *Drag the icon over the Trashcan, and release the selection button.*

If you open the Trashcan, the icon will be in the Trashcan window.

throwing away
the Copy_of.Expansion
drawer



2. *Make sure the Trashcan icon is selected (the lid will be open), then choose Empty Trash from the Icons menu.*

If you open the Trashcan window, the icon will be gone.

When you put an icon in the Trashcan, it will stay in the Trashcan until you choose Empty Trash from the Icons menu. As long as you have not selected Empty Trash, you can retrieve an icon from the Trashcan. To retrieve an icon, open the Trashcan window, and drag the icon into any window.

Some special rules apply to the Trashcan:

- You cannot delete a disk by dragging its icon over the Trashcan icon.
- You cannot move the Trashcan into a drawer.
- You cannot delete the Trashcan.

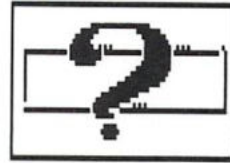
The Tools Menu

The Tools menu allows you to run application software by choosing a menu item instead of opening the program icon. If the application software supports this feature, there will be instructions on how to create the new menu item in the documentation accompanying the program.

Chapter 3. Preferences

In Chapter 2 you learned about the Workbench system and how the various components work. You should now be comfortable with the basic steps involved in using the Amiga, such as selecting and opening icons, choosing menu items, and working with windows.

Now it's time to put that knowledge to work. This chapter explains the **Preferences editors**. These editors, which are found in the Prefs drawer, let you personalize your Amiga environment. For instance, you can:



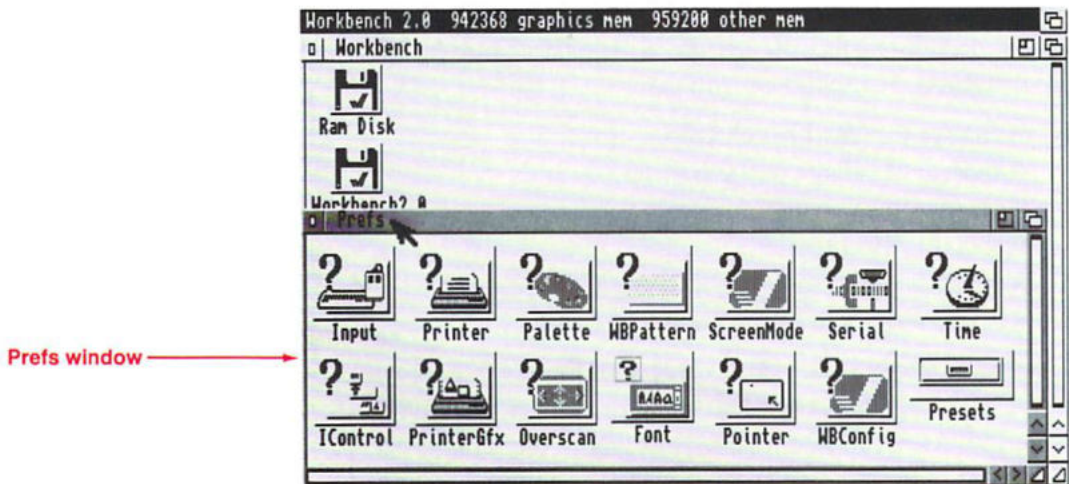
- change the Workbench colors
- change the shape of the pointer
- change the size of your display area
- specify your printer
- set up a printer for graphic output
- set up a modem for use with the Amiga

You do not need to use every editor in order to use your Amiga. Many of the editors let you change how things appear on the screen to suit your own tastes. However, if you connect a printer, modem, Multiscan or A2024 monitor to the system, you must use the appropriate editor to tell the Amiga how to interact with the new device.

After your system is set up the way you like it, you can move on to Chapter 4 and learn about the other programs included on the Workbench2.0 disk.

The Prefs Drawer

Open the Prefs drawer in the Workbench2.0 disk window, and the following window appears:



This window contains the icons for the Preferences editors, which are listed below in the order in which they are covered:

Time	Lets you set the date and time.
Input	Lets you change the mouse speeds (how fast the pointer moves, the length of time allowed for a double-click) and key repeat speeds.
Palette	Lets you change the colors of the Workbench.
WBPatten	Lets you select or create a background pattern for the Workbench screen and/or the Workbench windows.
Pointer	Lets you change the size, shape, and color of the pointer.

Font	Lets you change the fonts used in the different areas of the screen.
ScreenMode	Lets you choose a different display mode . This is necessary if you are using certain types of monitors with your system, such as an A2024 or Multiscan monitor.
Overscan	Lets you adjust the size of the display area for text and for graphics.
Printer	Lets you specify the printer driver that matches your printer and allows you to specify options such as paper size and margin width.
PrinterGfx	Lets you set up your printer to print graphics.
Serial	Lets you set the specifications for the serial port. This is used for communicating through modems or networking systems.
IControl	Lets you choose the keys used for some of the keyboard shortcuts, such as moving the Workbench screen or choosing an action gadget from a requester.
WBConfig	Lets you control whether icons appear on the backdrop or in the Workbench window. It also allows you to move a Workbench window to the front of the screen by double-clicking in the window.

All of the editors, except for Time, have menus which allow you to save different configurations for each editor. For instance, if you use two printers with your Amiga, you could save the settings for each printer in the Presets drawer. When you wanted to switch printers you could just open the appropriate file, instead of reselecting each individual printing option.

The menus and the Presets drawer, which are explained at the end of the chapter, are optional and do not have to be used in order to set up your Amiga.

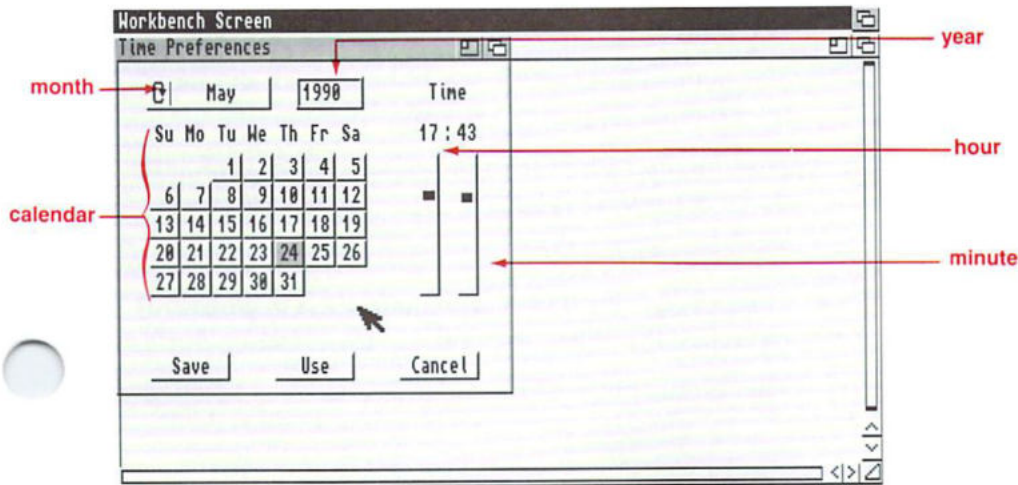
Action Gadgets

When you open an editor icon, a window appears on the screen. Each editor window has three action gadgets: Save, Use and Cancel. You must select one of these gadgets to implement your changes or to close the editor window. Their functions are explained below:

- | | |
|--------|---|
| Save | If you've made changes to the settings that you want to keep, select the Save gadget. This will implement any changes you have made, save the changes to disk and close the editor. The new settings will remain in effect even if you reboot the Amiga. |
| Use | If you've made changes to the settings that you want to try, <i>but you do not want to save them at this time</i> , select the Use gadget. This will temporarily implement any changes made and close the editor. If you reboot the Amiga, your changes will be lost, and the previously saved settings will be used. |
| Cancel | If you've made changes in the editor but decide you don't want to use them, select the Cancel gadget. This will close the editor without using or saving any changes made in the window. The settings that were in effect prior to opening the editor will remain in effect. |

The Time Editor

Open the Time icon, and the following window appears:



This window lets you set the correct date and time.

To set the date:

1. *Select the cycle gadget in the upper left corner until the correct month is displayed.*
As you select the gadget, the calendar will change to reflect the displayed month and year.
2. *If the incorrect year is displayed, select the year gadget, delete the incorrect information, and enter the current year.*
Click in the gadget, press Backspace or Del to erase the displayed year, type in the current year, and press Return.
3. *Select the correct day from the calendar display.*
The currently selected day will be highlighted.

To set the time:

1. *Point to the slider bar in the hour slider, and hold down the selection button.*

The bar is highlighted.

2. *Drag the bar until the correct hour is displayed at the top of the slider, then release the selection button.*

The hour value is based on a 24-hour clock and ranges from 0 (midnight) to 23 (11 PM).

3. *Point to the slider bar in the minute slider, and hold down the selection button.*

The bar is highlighted.

4. *Drag the bar until the correct minute is displayed at the top of the slider, then release the selection button.*

The minute range is from 00 to 59.

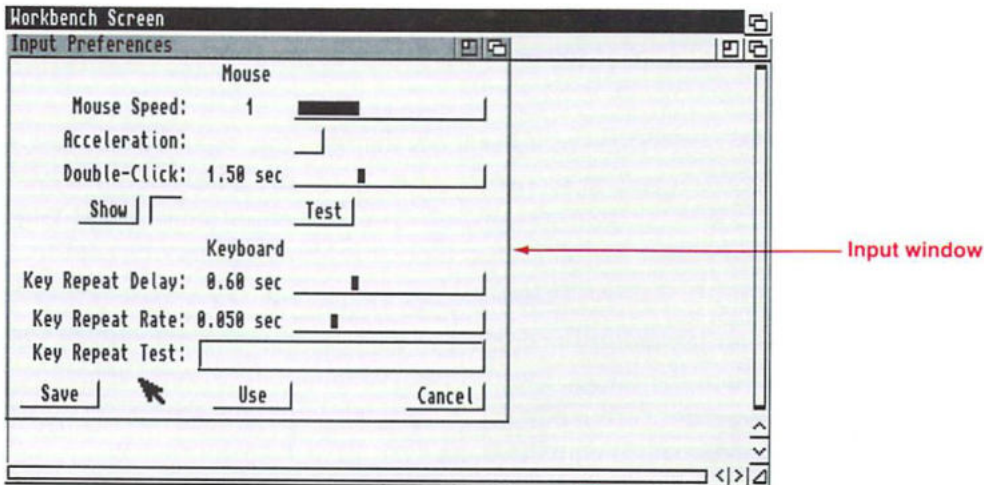
Save/Use/Cancel

To determine if your Amiga has a battery backed-up clock, refer to your Introducing the Amiga manual.

To save the displayed date and time to both the system clock and to the battery backed-up clock (if your Amiga has one), select the Save gadget. To only save the displayed date and time to the system clock, select the Use gadget. If the Amiga is rebooted or turned off, the date and time will be lost. To exit the editor without making any changes, select the Cancel gadget.

The Input Editor

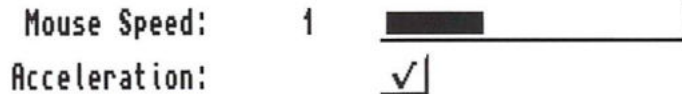
Open the Input icon, and the following window appears:



This window lets you change the speed at which the mouse and keyboard operate. There are two mouse speeds that you can change: how fast the pointer moves across the screen and how fast the mouse registers a double-click of the selection button. For the keyboard, you can change how fast a key repeats when it is held down continuously and how long it takes before a key will start to repeat.

Mouse Speed

The Mouse Speed slider lets you determine how fast the pointer moves across the screen as you move the mouse.



There are three mouse speeds that you can choose from: 1, 2, and 4. A setting of 1 is the fastest; 4 is the slowest. You may want to try each setting until you are comfortable. While you don't want the pointer to move too slowly, it may be hard to control if it is moving too fast.

Another factor to consider is the size of your work surface. When the mouse speed is fast, you don't need to move the mouse very much to get the pointer across the screen. A slower mouse speed requires more desk space for your mouse.

To set the mouse speed:

- 1. Point to the slider bar, and hold down the selection button.***

The bar is highlighted.

- 2. Drag the bar across the slider.***

When the bar is all the way to the left, the mouse speed is set to one. If the bar is all the way to the right, the mouse speed is set to four, and the pointer will move *significantly* slower.

- 3. When you reach the desired value, release the mouse button.***

As you drag the slider, the displayed mouse speed takes effect. This allows you to try out each speed without exiting the editor.

Acceleration

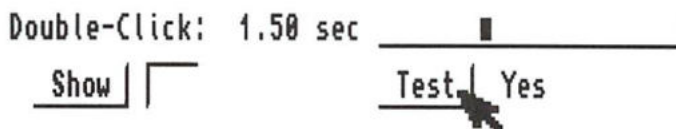
There may be times when you need to be able to move the mouse over large screen areas while keeping fine control for smaller movements. For instance, if you are using the mouse to create a complex drawing with a paint program, you may need to cover a lot of screen surface quickly but with precision.

If you simply increase the mouse speed, you lose a certain amount of mouse control. However, when **acceleration** is turned on, the mouse speed remains constant when you first start to move the pointer, allowing you to work precisely within a small area. However, as you move the pointer further across the screen, the mouse speed gets faster—similar to pressing the accelerator on a car. This allows you to cover large areas of the screen quickly.

When acceleration is on, a check mark appears in the check box. To turn acceleration on or off, select the check box.

Double-Click

The Double-Click slider determines the maximum amount of time allowed between the two clicks of a double-click.




The range is from 0.20 to 4.00 seconds. If you set the double-click speed to 0.20 seconds, you must click the mouse twice within two-tenths of a second in order to register a double-click. If you set the double-click speed to 4.00 seconds, you have a full 4 seconds between the two clicks of a double-click.

To see the amount of time for the selected value, select the Show gadget. A box appears next to the gadget and remains there for the length of time allowable between the two clicks.

To test the speed you've chosen, double-click in the Test gadget. If the double-click took place within the allotted time, a Yes appears next to the Text gadget. If there was too much time between your two clicks, a No appears.

Key Repeat Delay

Most of the keys on the keyboard automatically repeat when held down. However, there is a delay that occurs before the key starts repeating. The delay, measured in seconds, can be changed by using the Key Repeat Delay slider.


Key Repeat Delay: 0.60 sec 

You can see the current setting by looking at the slider value. The range is from 0.20 (shortest) to 1.50 (longest) seconds. For instance, if the current setting is 0.80 seconds, it will take eight-tenths of a second before the key starts to repeat.

To increase the delay before the key starts repeating, drag the slider bar to the right. To decrease the delay, drag the slider bar to the left.

Key Repeat Rate

The Key Repeat Rate slider determines the rate at which the keys will repeat, after the initial key repeat delay.

Key Repeat Rate: 0.050 sec 

The allowable range is from 0.002 (fastest) to 0.250 (slowest) seconds. To increase the speed, drag the slider bar to the left. To decrease the speed at which the keys repeat, drag the slider bar to the right.

Key Repeat Test

You can test the settings you have chosen for key repeat delay and key repeat rate with the Key Repeat Test gadget.

Key Repeat Test: 

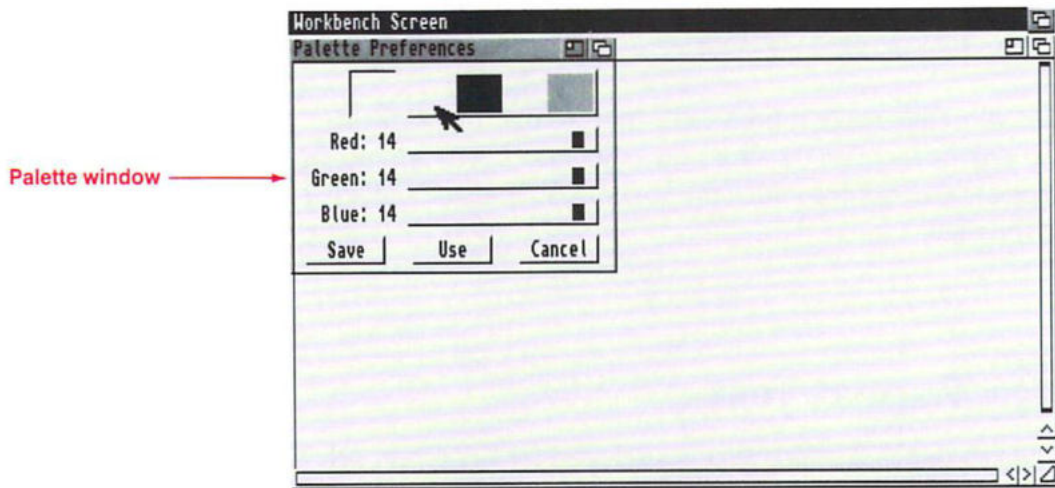
Select the gadget by pointing to it and clicking the selection button. A cursor will appear in the gadget. Hold down one of the letter keys on the keyboard. Your input will appear in the Key Repeat Test gadget and will show the current key delay and key repeat rates.

Save/Use/Cancel

To save your changes, select the Save gadget. If you only want to try your changes, select the Use gadget. To exit the Input editor without making any changes, select the Cancel gadget.

The Palette Editor

Open the Palette icon and the following window appears:



This window lets you change the colors of the Workbench. Each color on the Workbench screen is made up of various amounts of red, green, and blue. The red, green, and blue sliders, let you change the amounts of each color and create new colors. The current Workbench colors are shown in the selection gadget at the top of the window.

The number of available colors is determined by the type of display mode you are using and is set via the ScreenMode editor (explained later in this chapter). For instance, although the standard Hires display uses 4 colors, you can change the screen mode and use up to 16 colors. In this case the Palette editor would have 16 colors in its selection gadget.

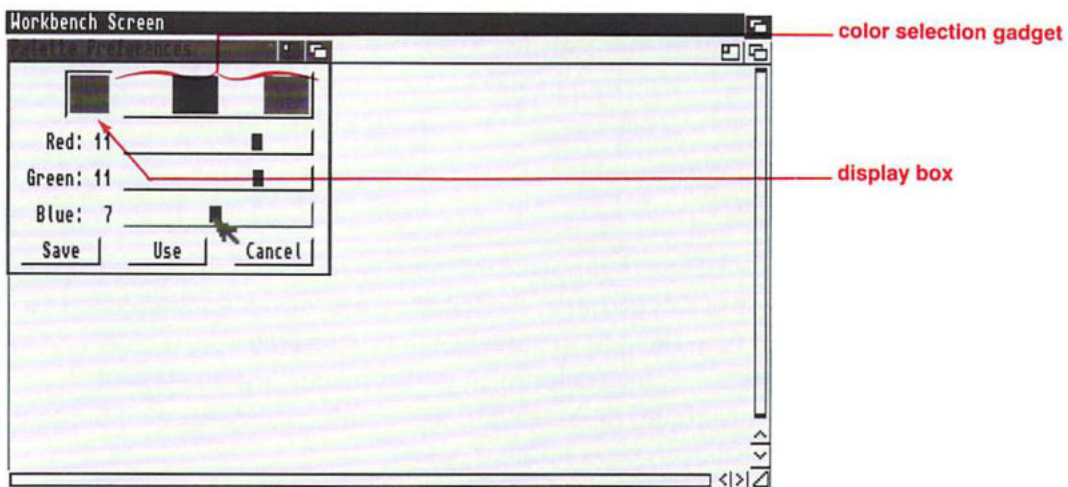
To change a color:

1. Select the color you want to change.

Point to a color in the selection gadget, and click the selection button. The selected color will be shown in the display box to the left of the selection gadget.

2. Use the slider gadgets to change the selected color.

By changing the values of the sliders you can change the amount of red, blue and green used in the selected color. Point to the slider bar in either the red, green or blue slider, hold down the selection button, and use the mouse to drag the slider bar. As you drag the bar through the slider box, the color in the display box and on the screen will change.



Be careful when changing the black and white settings as it could affect the three dimensional appearance of the screen. To preserve the three-dimensional effect, be sure to keep the color that replaces black darker than the color that replaces white.

By experimenting and changing the values for each of the three color sliders, you can create any of the Amiga's 4,096 colors. The table below shows some settings that will bring you close to your desired color. Remember these settings are just approximate. You may have to adjust the settings to achieve the color you want, as the actual shade of the color may vary depending on your monitor.

≡≡≡ Palette Color Table ≡≡≡			
Color	Red	Green	Blue
White	15	15	15
Yellow	15	15	0
Orange	15	7	0
Red	15	0	0
Brown	8	5	0
Gray	8	8	8
Purple	7	0	10
Lt. Blue	6	8	11
Green	0	8	0
Blue	0	0	15
Black	0	0	0

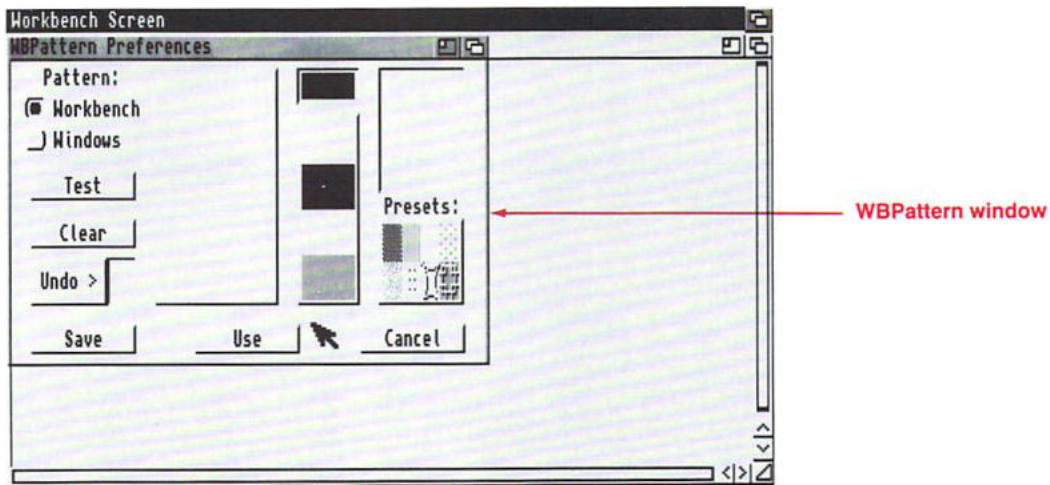
In addition to the standard Preferences menus (explained at the end of this chapter), the Palette editor also has a Presets menu item in the Edit menu which lets you choose from nine predetermined color settings. For instance, if you choose Sunset from the submenu, the default grey changes to blue and the default blue becomes orange.

Save/Use/Cancel

To save the colors, select the Save gadget. If you only want to try the colors, select the Use gadget. To exit the Palette editor without making any changes, select the Cancel gadget.

The Workbench Pattern Editor

Open the WBPattern icon and the following window appears:

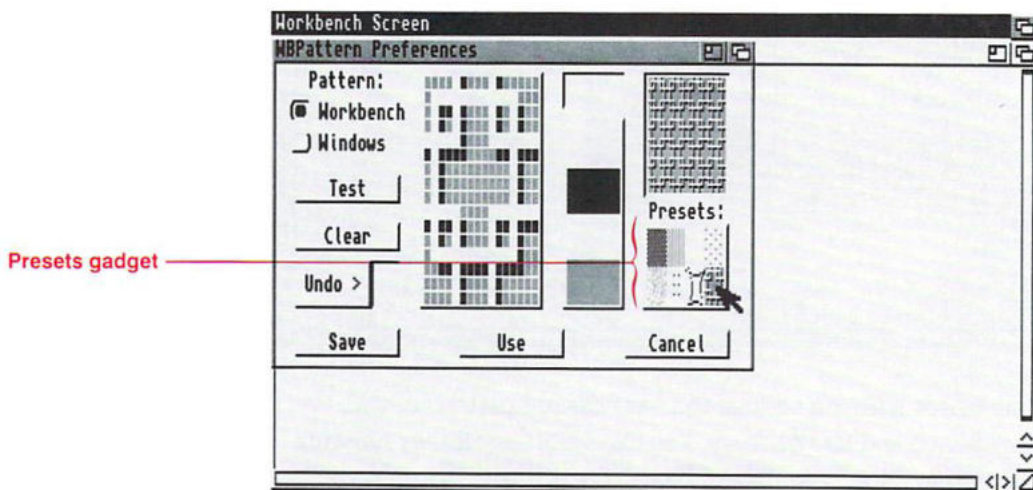


This editor lets you change the background pattern of the Workbench and its windows. You can select an already existing pattern or create your own. The pattern will fill any open areas of the window. The default is no pattern.

For a quick demonstration of how this works, follow the steps below:

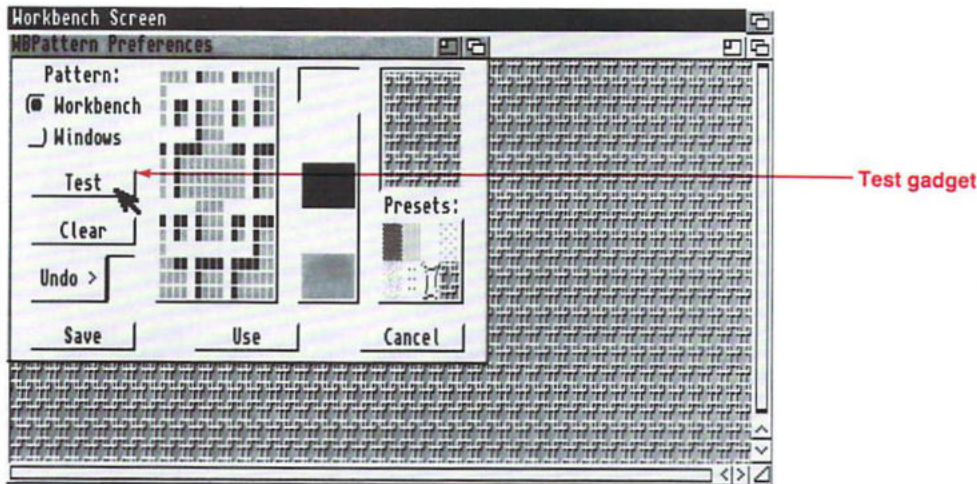
1. Select one of the patterns in the Presets gadget.

There are 8 patterns in the Presets gadget. Point to one, and click the selection button. A magnified view of the pattern will appear in the box in the middle of the window. An actual-size view will be shown in the display box above the Presets gadget.



2. Select the Test gadget.

The selected pattern will fill the background of the Workbench.



To create a pattern:

1. *Select the Workbench or Windows radio gadget.*

This determines whether the pattern appears in the Workbench window or any open disk or drawer windows.

2. *If there is a pattern in the magnified view box, select the Clear gadget.*

This will erase the contents of the magnified view box. The box will fill with the currently selected color.

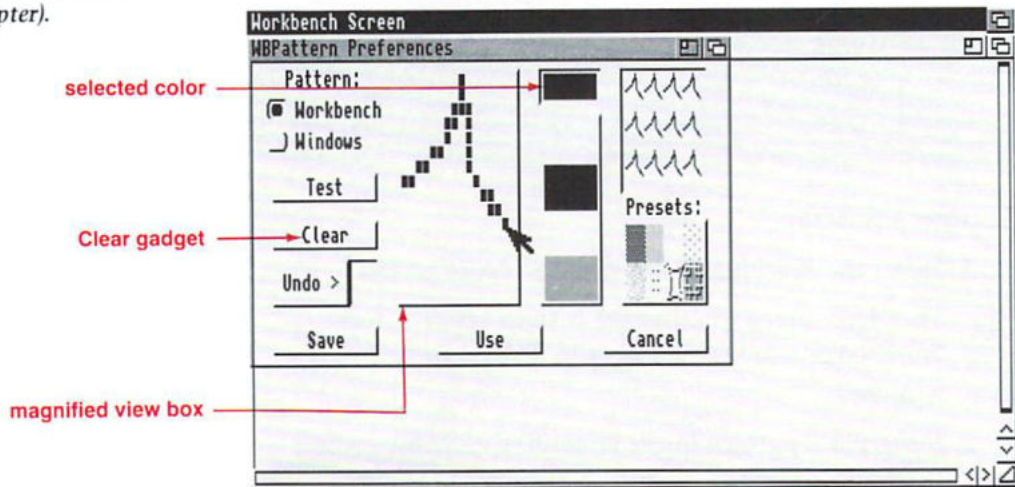
3. *Select a color to draw with from the color selection gadget.*

Point to a color in the selection gadget, and click the selection button. The selected color will appear in the display box above the gadget. You can also select a pattern from the Presets gadget, and then use the mouse to edit it.

Your screen is made up of rows and columns of tiny dots, or pixels. The number of pixels in a screen depends upon the display mode you have chosen (explained later in this chapter).

4. Point within the magnified view box to where you want to start drawing, then click the selection button.

One **pixel** of the selected color will appear. If you hold down the selection button and move the mouse, you can fill in several pixels at once. When you want to switch to another color, move the pointer over to the color selection gadget, and select the next color you want to use.

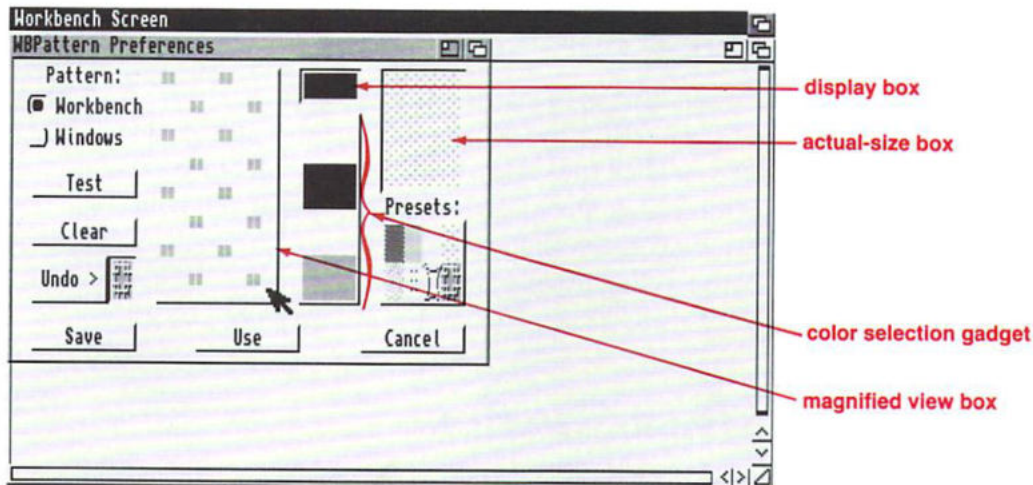


As you create your pattern in the magnified view, the pattern is repeated in the actual-size box. This gives you a better idea of how the pattern will look on the display.

5. When your pattern is finished, select the Test gadget.

The pattern will appear in the background of the Workbench window or any open disk/drawer windows, depending on which radio button you selected.

The rest of this section describes the various components of the Workbench Pattern editor.



Color Selection Gadget

The available colors are shown in a selection gadget to the right of the magnified view box. Select the color you want to use by pointing to it and clicking the selection button.

The selected color is shown in the display box above the color selection gadget. This is the color you will draw with when you click the selection button in the magnified view box.

When you want to draw with another color, move the pointer over to the selection gadget, point to a new color, then click the selection button.

The available colors are determined by the colors chosen with the Palette editor. You cannot change them within the Workbench Pattern editor. The number of colors is determined by the ScreenMode editor (explained later in this chapter).

Magnified View Box

You can use the mouse to draw a pattern inside the magnified view box. When the pointer is inside the magnified view box and you click the selection button, a small block of color appears. That block represents one pixel. If you hold down the selection button and move the mouse, you can draw with the mouse, filling in several pixels at once.

Actual-Size Box

This box shows the pattern in the size that it will appear on the Workbench. You cannot edit the pattern in this box.

Presets Gadget

There are eight preset patterns that you can select. When you select one of these patterns, it appears in both the magnified view and the actual-size boxes. You can then edit it in the magnified view if you wish.

Pattern

These buttons determine whether the pattern will appear on the Workbench or in any open disk/drawer windows.

Select the Workbench button, and the pattern that currently appears in the magnified view will be used on the Workbench. The pattern will not appear on the Workbench until you select the Test, Save or Use gadget.

Select the Windows button, and the pattern in the magnified view will be used in all disk and drawer windows.

You can switch back and forth between the two patterns by selecting the buttons. When you select the Workbench radio button, the last pattern that appeared in the magnified view box while the Workbench button was selected will reappear. When you select the Windows button, the pattern will change to the last pattern created for the windows.

Test

Select the Test gadget to try out the pattern. This gadget does not save the pattern. If you exit the editor without saving your pattern, the pattern will disappear.

Clear

Select the Clear gadget to erase the contents of the magnified view. The box will fill with the currently selected color.

Undo

Select the Undo gadget to cancel the last action performed by the mouse. The pattern in the Undo display area will be exchanged with the pattern in the magnified view box.

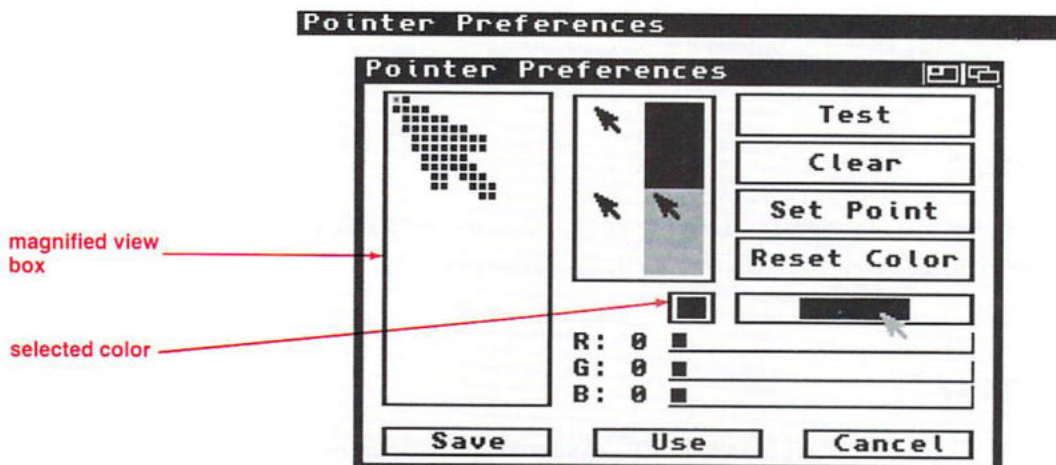
For instance, if you've just drawn a red line through the pattern, selecting Undo erases the red line (but leaves the rest of the pattern). However, if after drawing the red line, you accidentally clicked the selection button once and created one small dot of color in the box, selecting Undo will only erase the dot since that was the last action performed with the mouse.

Save/Use/Cancel

To save your patterns, select the Save gadget. If you only want to try the patterns, select the Use gadget. To exit the WBPattern editor without making any changes, select the Cancel gadget.

The Pointer Editor

Open the Pointer icon and a new screen appears:



This editor lets you change the size and shape of your pointer. A magnified view of the current pointer is shown in the left side of the window. It is this image that you modify to change the pointer. To the right of the magnified view are copies of the pointer which let you judge how the Pointer will look against the colors on the Workbench.

Although the pointer is only made up of three colors, there are four colors shown in the selection gadget. The left-most color is the Workbench background color and is transparent. This color cannot be changed. You will be able to see through any areas of the pointer drawn with this color. You can change the other three colors.

To change the colors:

- 1. Select the color you want to change from the color selection gadget.***

Point to the color and click the selection button. The selected color will be shown in the display box to the left of the selection gadget.

- 2. Change the amounts of red, green and blue in that color by using the three color sliders.***

Point to the slider bar in a color slider, and drag it to the left or right. The color in the display box and selection gadget will change as you do this so that you can see the color you are creating. Remember, you cannot change the leftmost color.

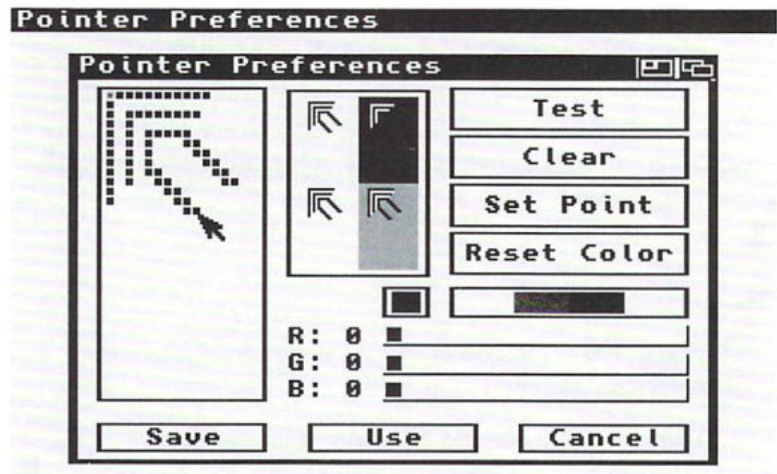
To edit the pointer:

- 1. Select the color with which you want to draw.***

Point to the color and click the selection button. The selected color will be shown in the display box to the left of the selection gadget.

- 2. Point to a place in the magnified view where you want to place a pixel of the selected color, then click the selection button.***

A small, colored rectangle appears where you've clicked the mouse. To draw several rectangles, hold down the selection button and move the mouse.



Repeat steps 1 and 2 to add other colors to your pointer. Just point to the color in the selection gadget that you want to draw with, and click the selection button.

The other gadgets on the screen are explained below:

Test

While you are drawing in the magnified view, the screen pointer does not change. The Test gadget lets you change the screen pointer so that it reflects what is drawn in the magnified view. This way you can see what the pointer looks like on the screen without closing the Pointer editor.

The pointer only changes at the time you select Test. If you select Test then keep editing the pointer, the pointer does not change to reflect the revisions. You must select Test again if you want to see any new changes to the pointer.

Clear

Select the Clear gadget to clear the magnified view. All pixels will be changed to the background color. You can then draw a new pointer.

Set Point

This gadget lets you determine where to put the pointer's point (or "hot spot"). The point is the single pixel in the pointer that must be over an icon in order to select the icon. In the magnified view, the point is indicated by a smaller square within one of the pixels.

To set the point:

1. *Select Set Point.*
2. *Point to the pixel in the magnified view where you want the point to be, then click the selection button.*

Reset Color

Select the Reset Color gadget to bring back the last set of colors that were saved.

Save/Use/Cancel

To save the pointer that is currently shown in the magnified view, select the Save gadget. If you only want to try the pointer, select the Use gadget. To exit the Pointer editor without making any changes, select the Cancel gadget. If you select Cancel after changing the screen pointer by selecting the Test gadget, a requester will ask if it is OK to discard the changes made to the pointer.

The Font Editor

A font is a set of characters of the same design.



The screen shown here represents a hard disk system. If you have a floppy disk system, only three sizes of the Topaz font will appear in the scroll gadget.

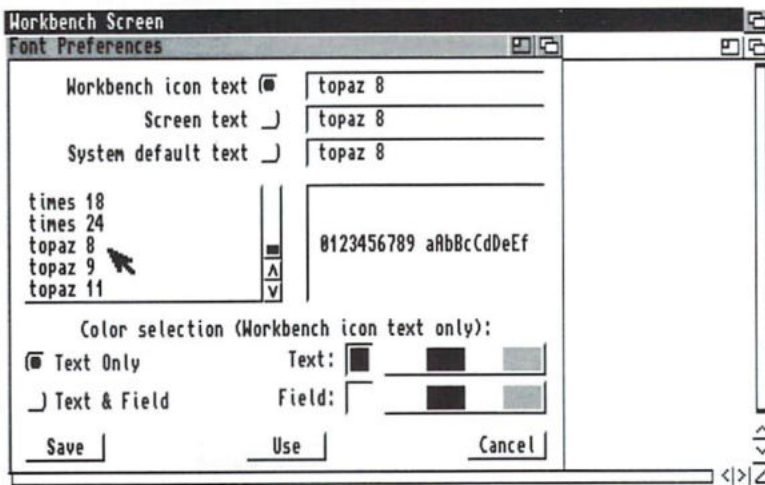
Font window →

The Font editor lets you change the **fonts** that are used on the Amiga.

The Font editor is primarily useful to people with hard disk systems. If you have a floppy disk system, the only font immediately available to you is Topaz, which is stored in the computer's **ROM** (Read-only memory). Additional fonts are supplied on the Extras disk, but to use these, you have to delete programs from your Workbench disk. This is not advisable unless you are an experienced Amiga user.

Never delete anything from your master copy of Workbench2.0.

Open the Font icon, and the following window appears:



This window lets you change the fonts that appear under the Workbench icons and in the Workbench menus and title bars. It also allows you to change the default system font that the Amiga uses to display information, such as the output of the View By menu item.

Text Radio Buttons

The radio buttons at the top of the Font editor window allow you to select the text that you want to change.

Workbench icon text	<input checked="" type="radio"/>	topaz 8
Screen text	<input type="radio"/>	topaz 8
System default text	<input type="radio"/>	topaz 8

Your choices are:

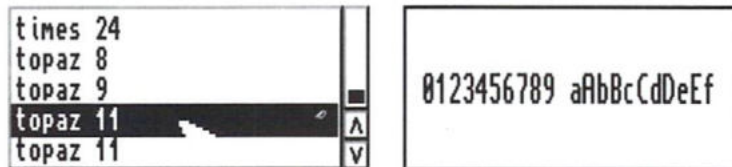
- | | |
|---------------------|--|
| Workbench icon text | Changes the font below icons in the Workbench windows. This is the only text for which you can specify a color (see the "Text/Field" section below). Any changes to the Workbench icon text will take effect when you exit the Font editor. |
| Screen text | Changes the font that appears in all screens — in the menus, the title bar, requesters, etc. This font will only take effect when the Workbench is reset. When you select the Save or Use action gadget, the Font editor will close, and the Workbench will automatically try to reset. If you have any project or tool windows open, a requester will ask you to close them. You can leave disk or drawer windows open. |
| System default text | Changes the font that the Amiga uses to display information, such as the text in the Workbench Output Window. This font change is not immediately noticeable. |

To select a text area, point to the radio button next to that area, and click the selection button. Remember, you can only change the font for one text area at a time. However, you can change the font for each area without exiting the Font editor. For instance, you can select the Workbench icon text radio button, and select a font for the icons. Then, select the Screen text radio button, and select a font for the screen. Finally, select the System default text radio button, and select a font for that area.

NOTE: Many application programs choose their own fonts and are not affected by your choices in the Font editor.

Font Gadget

When you select a radio button, a list of available fonts appears in the scroll gadget. The font name is followed by a number. The number represents the size of the font—the higher the number, the larger the font. Sizes vary from font to font, as some fonts are naturally smaller than others. The maximum size allowable is 124 points.



A nonproportional font is one where every character is the same width.

When you are selecting a font for the Workbench icon text or Screen text, all of the fonts are available. However, the font used for the System default text must be a **nonproportional font**, such as topaz or courier.

To see the names of the available fonts, drag the scroll bar, or use the scroll arrows, to scroll through the fonts. To select a font, point to it and click the selection button. The name of the selected font will be shown in the text gadget next to the selected radio button.

An example of the chosen font, in its actual size, is shown in the display box next to the font scroll gadget. If you do not like the way a font looks, select another one. You can keep making selections as long as the appropriate radio button remains selected.

Text/Field

NOTE: This is only applicable to the Workbench icon text.

When changing the font for the Workbench icons, you can also specify the color of the text and the **field**. Two radio buttons let you choose between Text Only or Text & Field.

The field is the area immediately surrounding the text.

Color selection (Workbench icon text only):

<input checked="" type="radio"/> Text Only	Text:	
<input type="radio"/> Text & Field	Field:	

If you select the Text Only radio button, the text will be the color specified by the Text color selection gadget. The field will be the Workbench background color or pattern.

If you select the Text & Field radio button, you can specify a color for both the text and the field. This option ensures that the text is legible regardless of the background pattern.

To change the color of the text, select a color from the Text color selection gadget by pointing to the color of your choice and clicking the selection button. To change the color of the field, select a color from the Field color selection gadget. *Be sure to select two different colors for the text and the field. Otherwise, the text will blend in with the field, and you will not be able to read the words.*

The number of available colors is determined by the ScreenMode editor, while the colors themselves are determined by the Palette editor. You cannot change the colors within the Font editor.

Save/Use/Cancel

To save the fonts shown in the text gadgets, select the Save gadget. If you only want to try the fonts, select the Use gadget. To exit the Font editor without making any changes, select the Cancel gadget.

Remember, if you've changed the Screen text font, the Amiga will attempt to reset the Workbench screen. All tool or project windows must be closed.

Types of Displays

This section explains the types of displays that you can use with your Amiga so that you will be aware of your choices before using the ScreenMode or Overscan editors.

The ScreenMode editor lets you change the display mode for the Workbench screen. The display mode refers to the number of horizontal pixels and the number of vertical pixels in a screen. This is also known as the screen **resolution**.

The standard Workbench screen that appears when you boot with your original Workbench disk is a Hires (high-resolution) screen. It is 640 pixels wide (left to right). Its height is determined by your country's video standard. For an NTSC display, a Hires screen is 200 pixels high (top to bottom); for a PAL display it is 256 pixels high.

Most display modes provide an **interlaced** option which doubles the number of horizontal lines on a screen, thereby increasing the resolution. Depending on which model of Amiga you own, your system may use a Hires-Interlaced screen (400 lines NTSC; 512 lines PAL). Interlaced screens may flicker when used with certain monitors. Some Amiga models have special hardware installed to eliminate the flicker when used with the appropriate monitor. It is possible to add similar hardware to other Amiga models as well.

The display modes available to you may depend upon the type of monitor you are using. Each display mode is explained in the following sections and the chart on page 3-32 lists all the display modes, the hardware needed to use that mode, and the standard screen sizes.

Remember that the display mode you choose only pertains to the Workbench screen. If an application opens its own screen, you should check the documentation supplied with the software to see which display modes the application supports.

In the following sections describing the various display modes, the information along the right margin is presented in the following format:

- width × height of standard screen
- width × height of interlaced screen
- maximum number of colors

Height is shown for both NTSC and PAL systems—NTSC/
PAL.

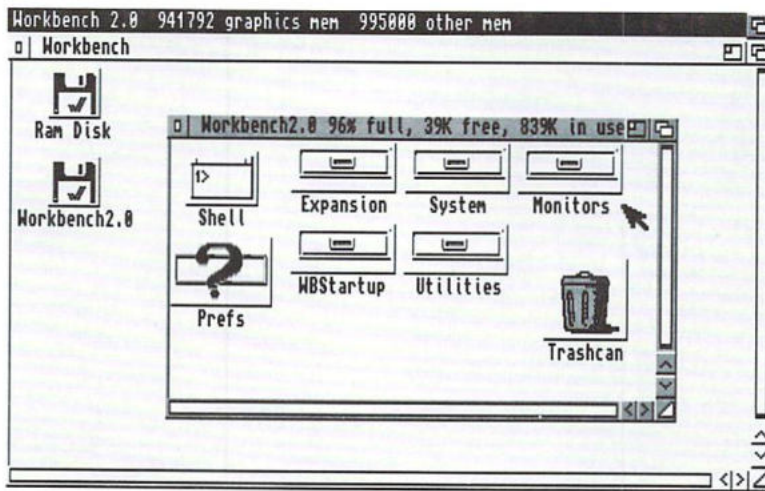
Possible Display Modes		
Display Mode	Special Requirements	Standard Screen Size
NTSC: Hires	NTSC monitor ¹	640 x 200
NTSC: Hires Interlaced	NTSC monitor ¹	640 x 400
NTSC: SuperHires	NTSC monitor ¹	1280 x 200
NTSC: SuperHires Interlaced	NTSC monitor ¹	1280 x 400
PAL: Hires	PAL monitor ¹	640 x 256
PAL: Hires Interlaced	PAL monitor ¹	640 x 512
PAL: SuperHires	PAL monitor ¹	1280 x 256
PAL: SuperHires Interlaced	PAL monitor ¹	1280 x 512
Productivity	Multiscan monitor ²	640 x 480
Productivity Interlaced	Multiscan monitor ²	640 x 960
A2024_10Hz	A2024 monitor ²	1008 x 800
A2024_15Hz	A2024 monitor ²	1008 x 800

¹If this is not the standard video mode for your country, it will only be available if you have dragged the appropriate monitor from the MonitorStore drawer to the Monitors drawer.

²You must drag the appropriate monitor from the MonitorStore drawer to the Monitors drawer in order for this monitor to be recognized by the Amiga.

Hires

640 × 200/256 non-interlaced
640 × 400/512 interlaced
16 colors maximum

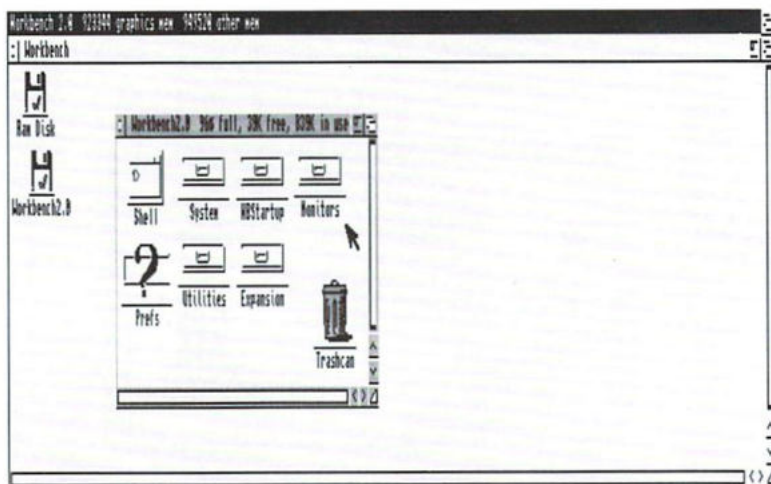


This is the default display mode used by the Amiga. It is suitable for most text-based applications, such as word processing and databases.

Unless you have display enhancer hardware, the Hires-Interlaced screen may flicker.

SuperHires

1280 × 200/256 non-interlaced
1280 × 400/512 interlaced
4 colors maximum



A SuperHires screen essentially cuts the width of the pixels used by a Hires screen in half. It doubles the amount of information that can fit on the screen, making text and icons considerably smaller. SuperHires may be especially useful for video applications.

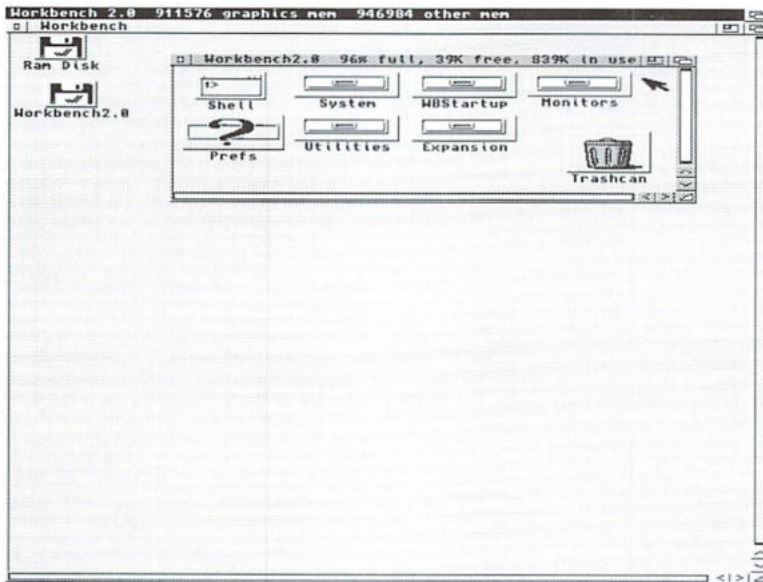
SuperHires is only available on machines equipped with the **Enhanced Chip Set**. If you have display enhancer hardware, you should disable it when using SuperHires mode, or the display may be distorted. (See the display enhancer documentation for instructions.)

Productivity

640 × 480 non-interlaced

640 × 960 interlaced

4 colors maximum



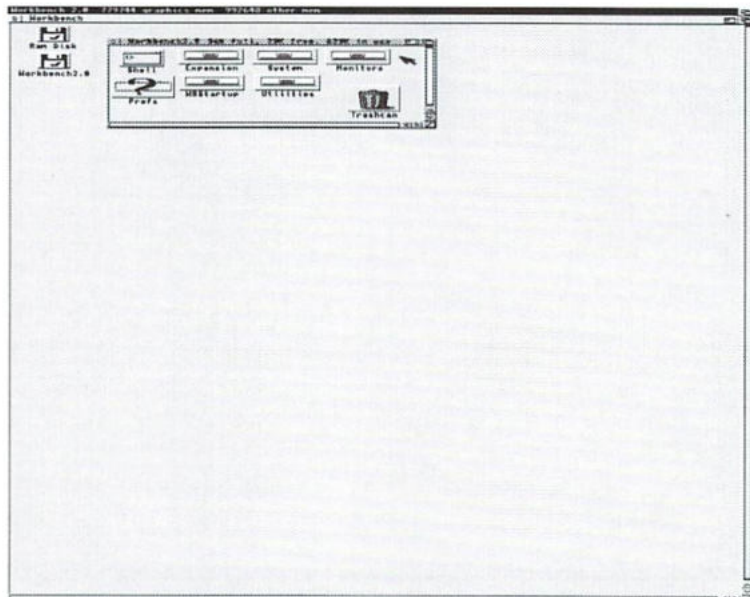
Productivity mode looks similar to the Hires-Interlaced mode. You must have the Enhanced Chip Set and a Multiscan monitor in order to use Productivity mode.

If you do not have display enhancer hardware, you can use Productivity mode to display 480 vertical lines without any flickering or visible scan lines. This is useful for desktop publishing, CAD/CAM, and graphics programs.

If you do have display enhancer hardware, a Hires-Interlaced screen is a better choice as it uses fewer system resources than Productivity mode while providing more colors.

A2024

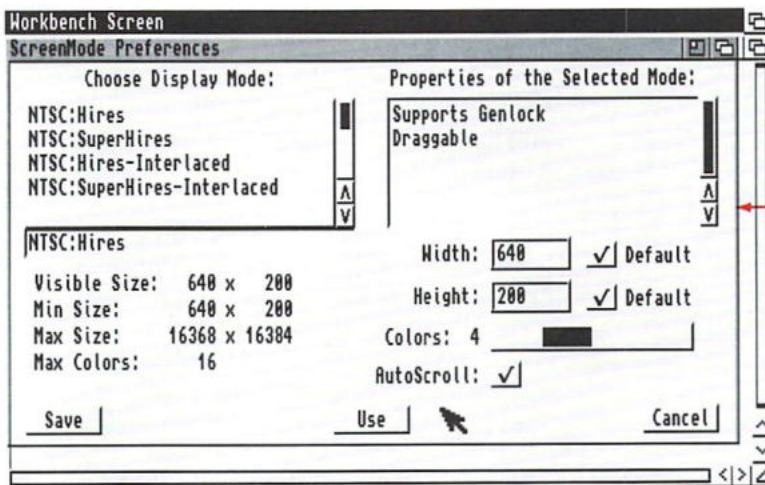
1008 × 800 only
no interlaced option
4 shades of grey maximum



The A2024 mode is only available on an Amiga with an A2024-style monochrome monitor. These display modes are commonly used for desktop publishing and CAD/CAM programs as they allow you to display a complete 8.5 × 11 inch page on the monitor screen. The 10Hz mode is recommended for text editing. The 15Hz mode refreshes the screen more frequently, providing a better picture. However it uses more system resources than the 10Hz mode.

The Screen Mode Editor

Open the ScreenMode icon and the following window appears:



ScreenMode window

This window lets you select the display mode for the Workbench screen. The different modes were explained in the previous section.

Choose Display Mode

The available display modes are shown in the Choose Display Mode scroll gadget. If there are several modes available, you may have to scroll through the list to see all of the options.

To select a display mode from the list, point to it and click the selection button. The selected mode will appear in the display box under the Display Mode gadget.

Properties of the Selected Mode

This display box lists information about the display mode you selected. The possible properties, depending on the selected mode and the icons in the Monitors drawer, include:

Interlaced	This shows whether or not the display mode supports an interlaced screen.
ECS	Certain display modes are only available if you have the Enhanced Chip Set in your Amiga.
PAL	If an NTSC machine has the Enhanced Chip Set and the PAL icon is in the Monitors drawer, PAL display modes will be available.
NTSC	If a PAL machine has the Enhanced Chip Set and the NTSC icon is in the Monitors drawer, NTSC screens will be available.
Supports Genlock	This shows whether or not the display mode supports the use of genlocking equipment.
Draggable	This shows whether or not the display mode supports a draggable Workbench screen. A draggable screen can be pulled down to reveal any other open screens behind it.
Panelled	This appears when the selected display mode is made up of several panels, such as the A2024 screen.
Requires bypassing the Display Enhancer	This indicates that display enhancer hardware should be disabled when using this display mode.

Screen Sizes

Various screen measurements are shown in the lower, left corner of the window. The size is given in the number of pixels. The first number represents the width of the screen; the second number represents the height.

```
Visible Size:  640 x  200
Min Size:      640 x  200
Max Size:      16368 x 16384
Max Colors:    16
```

The sizes correspond to the currently selected display mode.

- | | |
|--------------|---|
| Visible Size | This reflects the size of the text overscan area, as determined by the Overscan editor (explained in the next section). |
| Min Size | This is the smallest, or minimum, screen size that the selected display mode supports. |
| Max Size | This is the largest, or maximum, screen size that the selected display mode supports. The amount of chip memory available may further restrict this size. |

NOTE: If you try to use a screen size that is beyond the capability of the available graphics memory, the system will default to the screen size currently selected.

- | | |
|------------|--|
| Max Colors | This is the maximum number of colors that can be displayed on a screen in the selected display mode. |
|------------|--|

Width

Use the Width text gadget to specify the width of your Workbench screen.

Width: ☒ Default

Enter a number between, or equal to, the minimum and maximum widths. Once you enter a number, that number remains constant *no matter which display mode you select until you select the Default gadget*. However, if the number you've entered is larger than the maximum size for a selected display mode, the width value decreases to the maximum size.

The Default check box to the right of the Width gadget, allows you to select the default setting for width. This is usually equal to the width shown for the visible size measurement.

Height

Use the Height text gadget to specify the height of your Workbench screen.

Height: ☒ Default

Enter a number between, or equal to, the minimum and maximum heights. Just as with the Width gadget, the height size remains constant regardless of the selected display mode until you select the Default gadget.

Colors

This slider gadget lets you select the number of colors that can be displayed on the screen.



To increase the number of colors, point to the slider bar, hold down the selection button, and drag the bar to the right. To reduce the number of colors, drag the bar to the left. The number of colors selected is shown to the left of the slider.

The fewer colors selected, the faster the screen can be redrawn. Fewer colors also use less memory.

AutoScroll

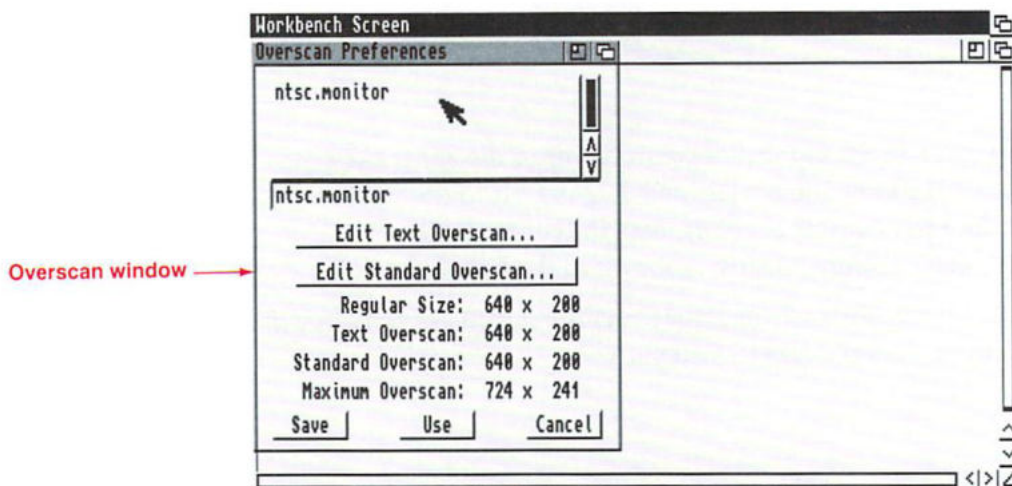
If you've specified a screen width that is larger than the monitor's display area, you may want to turn on the **AutoScroll** option. When AutoScroll is selected, the screen automatically starts to scroll when the mouse reaches the edge of the visible portion of the screen.

Save/Use/Cancel

To save the display mode, select the Save gadget. If you only want to try the settings, select the Use gadget. To exit the ScreenMode editor without making any changes, select the Cancel gadget.

The Overscan Editor

Open the Overscan icon and the following window appears:



Your screen usually fills most of the monitor's display area, but there is often a small amount of unused space around the edges of the monitor screen. This area is known as the **overscan** area. This window lets you enlarge the size of your screen so that you can take advantage of that unused space.

In general, any application software that uses a Workbench screen should support the sizes you select with the Overscan Editor. It is possible that some older software will have limits on the window size.

The various display groups for which you can enlarge the overscan areas are shown in the scroll gadget in the top of the window. As explained earlier, the number of display groups you can choose from depends on the type of chip set in your Amiga and, possibly, the type of monitor you are using.

The default group will be the video standard for your country, PAL or NTSC. If your monitor supports both video standards, and the appropriate icon is in the Monitors drawer, the other video standard will also be available. Do not try to use the other video standard unless your monitor supports it, or your screen image may be garbled.

The possible groups and the modes they represent are outlined below:

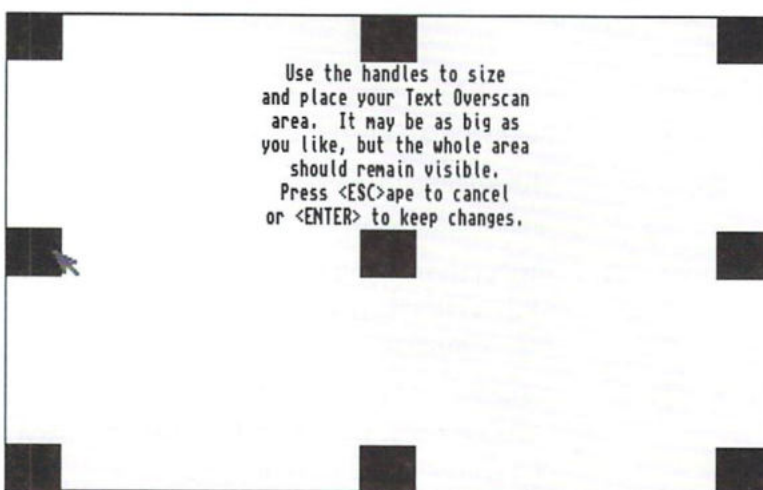
NTSC	Hires/Hires-Interlaced SuperHires/SuperHires-Interlaced
PAL	Hires/Hires-Interlaced SuperHires/SuperHires-Interlaced
Multiscan	Productivity/Productivity-Interlaced Only available if a Multiscan monitor has been connected to the system.
A2024	A2024 — 10Hz and 15Hz Only available if an A2024 monitor has been connected to the system.

When you change the overscan values for a particular display group, all modes within that group are affected. To change the overscan values for a display group, point to the group in the scrolling list and click the selection button. The selected group will appear in the display box under the scroll gadget.

Edit Text Overscan . . .

NOTE: This function has no effect when used with the A2024 display modes.

This gadget lets you adjust the area available for text display. When you select the gadget, the following screen appears:



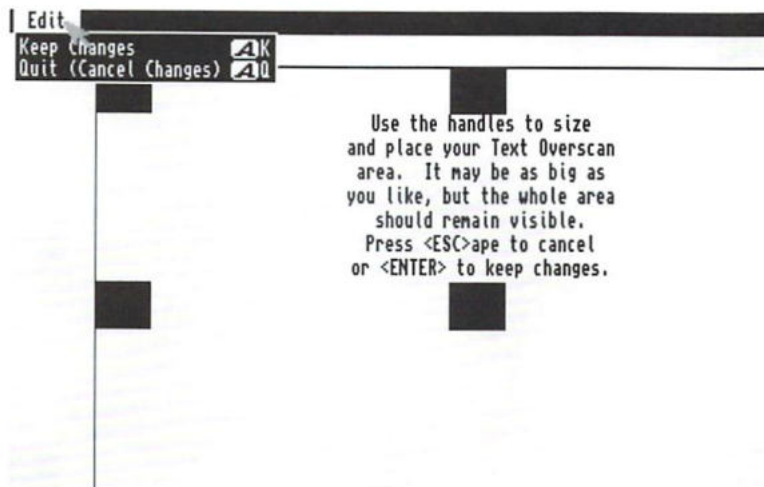
The black rectangles are handles. The handles, and the line connecting them, represent the outermost area where text can be displayed. To enlarge the overscan area, point to a handle, hold down the selection button, and drag the handle to the edge of the screen. Do this with the handles on each side of the screen.

Be careful not to move any part of the handle off the screen. If part of a handle is out of the viewing area, you will not be able to see all of the text on a screen.

Use the center handle to position the screen. By dragging the handle, you can shift the screen slightly to the right or left, or up or down. In this way you can center the display area on your monitor screen without having to use the monitor's horizontal and vertical controls.

To exit the screen without saving any changes, press Esc. To save changes, press Return. You will be returned to the Overscan editor.

You can also exit the screen with a menu. Hold down the menu button and point to the top left corner of the Text Overscan screen. A menu will appear.

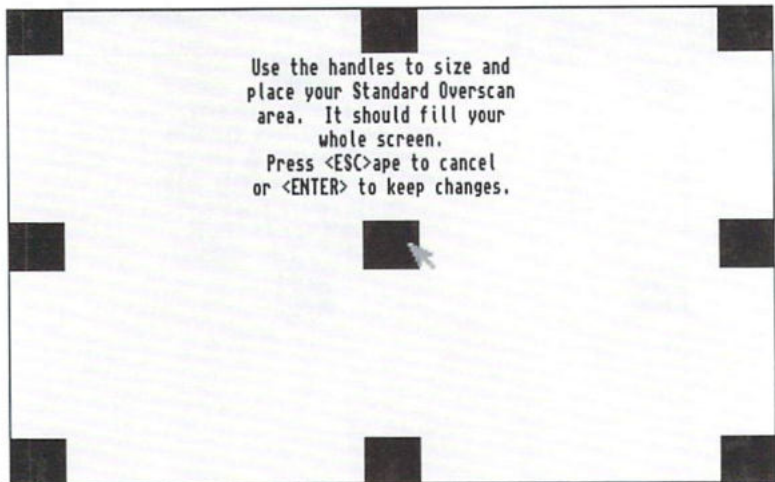


Choose Keep Changes (or press right Amiga-K or Return) to save your changes and exit the screen. Choosing Quit (right Amiga-Q) or pressing Esc allows you to leave the screen without implementing your changes. You will be returned to the Overscan editor.

Edit Standard Overscan . . .

NOTE: This function has no effect when used with the A2024 display modes.

This gadget lets you adjust the standard display size. When you select the gadget, the following screen appears:



The handles, and the line connecting them, represent the outermost area where data will be displayed. At times, you may want your graphics to fill as much of the screen as possible, even running off the screen so that there is no discernible border surrounding the picture.

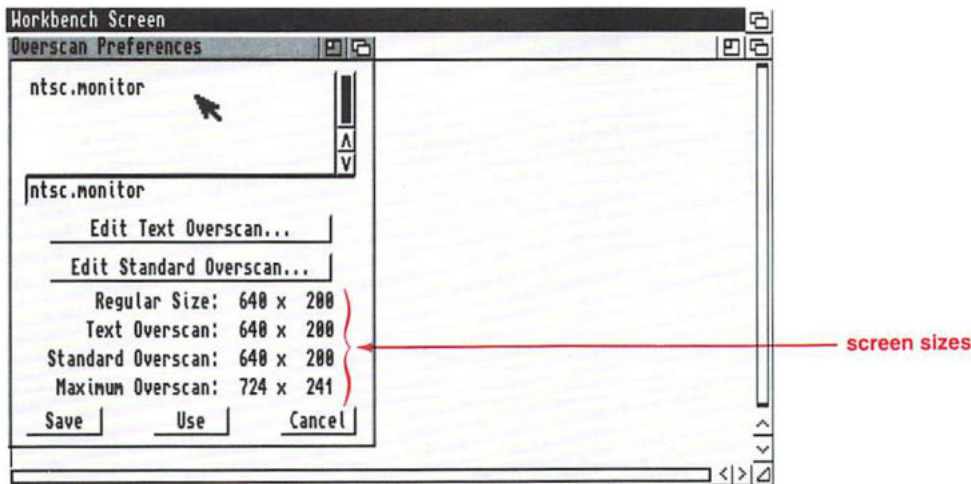
Use the mouse to drag the handles as much as necessary so that the line connecting them completely encompasses the screen. The resulting area should be slightly larger than the monitor screen. As with the Text Overscan Editing screen, you can use the center handle to position the screen.

To exit the screen without saving any changes, press Esc. To save changes, press Return. You will be returned to the Overscan editor.

You can also use the menu to exit the screen. Choose Keep Changes (or press right Amiga-K) to save your changes and exit the screen. Quit (or right Amiga-Q) allows you to leave the screen without implementing your changes. You will be returned to the Overscan editor.

Screen Sizes

The different sizes of your overscan areas are displayed at the bottom of the Overscan window.



The sizes are given in number of pixels. The first number represents the width of the screen (left to right); the second number represents the height of the screen (top to bottom).

The sizes correspond to the sizes for a standard Hires screen when editing the PAL or NTSC groups. If Multiscan is selected, the sizes for a non-interlaced Productivity screen will be given. All other display modes in a group will be affected proportionally.

The regular size is sometimes referred to as the nominal or standard size.

The different size categories are explained below:

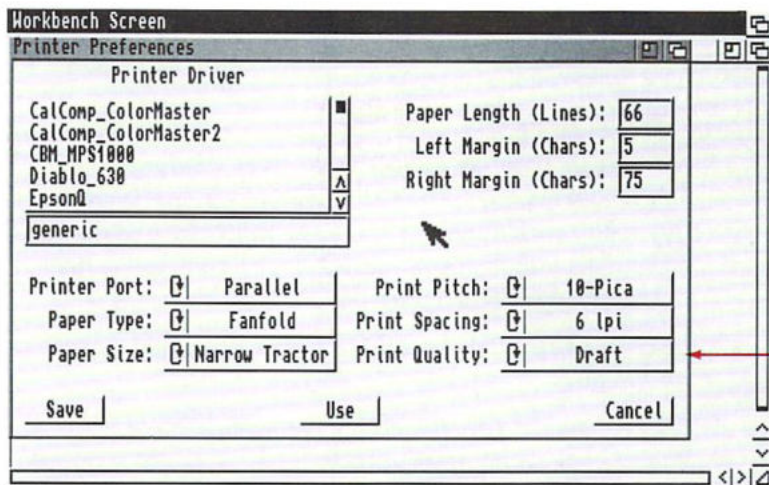
Regular Size	This is the standard, non-overscan size of a screen.
Text Overscan	This reflects the size of the current text overscan area. As you enlarge or reduce the text overscan area, this value changes.
Standard Overscan	This reflects the size of the current standard overscan area. As you enlarge or reduce the standard overscan area, this value changes.
Maximum Overscan	This is the maximum allowable size for any overscan area, text or standard.

Save/Use/Cancel

To save your new overscan sizes, select the Save gadget. If you only want to try the sizes, select the Use gadget. To exit the Overscan editor without making any changes, select the Cancel gadget.

The Printer Editor

Open the Printer icon and the following window appears:



If you have a floppy disk system, your window may look slightly different. It will not contain a list of available printer drivers in the scroll gadget.

This window lets you tell the system what type of printer you are using along with what type of output you want.

The first thing you need to do is to select a printer driver. A printer driver is software that enables the Amiga to communicate with a particular model of printer. The drivers are usually named for the printers they represent. The printer driver selected is used as both the text and graphic printer.

If you have a floppy disk system, the printer drivers are stored on the Extras2.0 disk. However, the Printer editor looks for the drivers on the Workbench2.0 disk. You must copy the driver for your printer to your Workbench2.0 disk. For instructions on how to do this, see the box on page 3-50. The driver will then appear in the Printer Driver scroll gadget. To select it, point to it and click the selection button. The selected driver is shown in the text gadget underneath the scroll gadget.

If you are unsure of which driver to select, please refer to Appendix B, "Printer Drivers." Some drivers are capable of supporting more than one printer.

Copying a Printer Driver from
Extras2.0 to Workbench2.0

1. *Select the Workbench2.0 window, then choose Show All Files from the Windows menu.*

2. *Double-click on the Devs drawer.*

You can close the Workbench2.0 window if you wish. This will keep the screen from getting cluttered with windows.

3. *Look for the Printers drawer in the Devs window.*

You do not need to open the Printers drawer. Just leave it in a visible part of the window. Once you've located the appropriate printer driver, you will copy it by dragging the driver icon over the Printers drawer icon.

4. *Insert the Extras2.0 disk into your disk drive, and open its window.*

5. *Select Show All Files from the Window menu, and double-click on the Devs drawer.*

You can safely close the Extras2.0 window if you like.

6. *Double-click on the Printers drawer in the Devs window.*

The Printers window will contain icons for the various printer drivers. Scroll through the window until you find the driver that works with your printer. (If you are unsure of which driver to use, see Appendix B for additional specifications.)

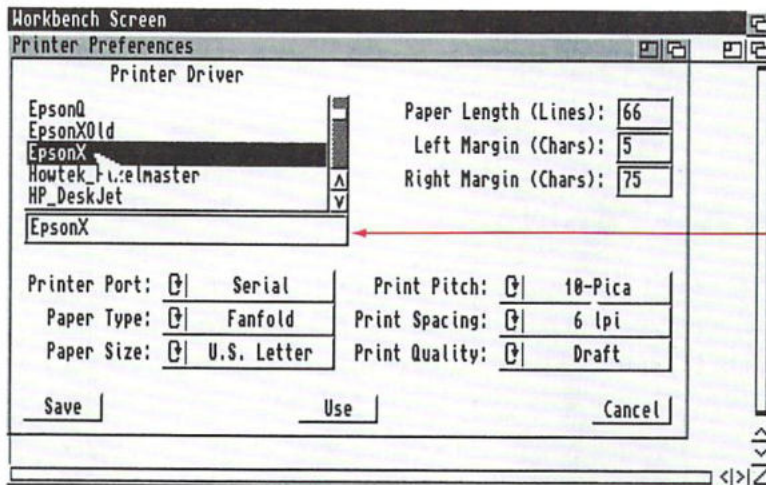
7. *Drag the appropriate printer driver icon over the Printers drawer icon in the Workbench2.0 Devs window.*

If you have a single-drive system, requesters will appear asking you to swap the Workbench2.0 and Extras2.0 disks until the printer driver is copied.

If you have two drives, put a disk in each drive, and the driver will be copied directly from one disk to the other.

The next time you open the Printer editor, the name of the printer driver will appear in the scroll gadget.

If you have a hard disk system, all the available drivers are shown in the scroll gadget in the upper left corner of the editor. Drag the scroll bar to see all the available drivers. When you see a driver for your printer, point to it and click the selection button. The selected printer driver is shown in the text gadget underneath the scroll gadget.



selected driver

If a driver for your printer is not in the list, check to see if a disk with an Amiga printer driver file was included with your printer. If the instructions for your printer tell you to indicate a printer file, select the text gadget and type in the name of the file included with your printer. (Be sure to specify the complete path to the file or copy the file into the Devs/Printers drawer on the Workbench2.0 disk.)

If you want to use a printer that is not included in the list, and you do not have a printer driver for it, enter generic in the text gadget. For many printers this allows you to print plain text, but not graphics or extra type styles, such as italics and boldface.

Once you have chosen your printer driver, the other gadgets in the window let you set the specifications for your printer's output.

NOTE: The specifications you set with this editor may be overridden when you use certain application packages, like desktop publishers or word processors. Those types of programs usually ask you to specify print specifications specifically for that program.

Paper Length

1 inch = 25.4mm

This gadget determines the total number of lines on your page, including top and bottom margins. For instance, if you are using 11 inch long paper, with 6 lines to an inch (this is set with another gadget), you will have 66 lines on your page.

To set the paper length, select the Paper Length text gadget, delete the current value, type in the correct value, and press Return.

Left Margin

1 inch = 25.4mm

This gadget determines the width of the left margin — the number of characters from the left edge of the paper to where you want text to start printing. For instance, if you want a one-inch margin, and you are using 10 **pitch** type, enter 10. (Pitch refers to the number of characters in a horizontal inch and is explained later in this section.)

To enter the value, select the Left Margin text gadget, delete the current value, type in a new value, and press Return.

Right Margin

This gadget determines the width of the right margin — the number of characters from the *left-hand edge of the paper* to where you want the right margin to begin. For example, if your paper is 8.5 inches wide, and you're using 10 characters per inch, you can fit 85 characters across a page (8.5×10). To leave a one-inch right margin, you must subtract 10 characters from the right edge of the page. The right margin would be 75.

To enter the value, select the Right Margin text gadget, delete the current value, type in a new value, and press Return.

1 inch = 25.4mm

Printer Port

This cycle gadget lets you specify the port where you have attached your printer. Your options are Parallel or Serial. The displayed option is the selected option.

If you need to change the displayed option, select the Printer Port gadget, and the other available option will appear.

Paper Type

This gadget specifies the type of paper you are using. The options are Fanfold (continuous-feed paper) or Single (individual sheets). The displayed option is the selected option.

If you need to change the displayed option, select the Paper Type gadget, and the other available option will appear.

1 inch = 25.4mm

Tractor-feed paper has holes along the side that attach to the sprockets on your printer.

Paper Size

This gadget specifies the size of paper you are using. You have seven choices:

Narrow Tractor	9.5 inches wide by 11 inches long (241 millimeters by 279 millimeters)
Wide Tractor	17.875 inches wide by 11 inches long (454 millimeters by 279 millimeters)
Custom	When you select Custom, you must be sure to specify the correct number of lines that fit on your paper. Do this through the Paper Length gadget described on page 3-52. Sometimes when printing in graphics mode on Epson and other dot matrix printers, narrow blank lines appear across the printout. Selecting Custom may eliminate this.
DIN A4	8.3 inches wide by 11.7 inches long (210 millimeters by 297 millimeters)
DIN A5	5.8 inches wide by 8.3 inches long (148 millimeters by 210 millimeters)
U.S. Letter	8.5 inches wide by 11 inches long (216 millimeters by 279 millimeters)
U.S. Legal	8.5 inches wide by 14 inches long (216 millimeters by 356 millimeters)

To make your choice, keep selecting the Paper Size gadget until the appropriate option is displayed.

Print Pitch

Pitch refers to the number of characters printed in 1 inch of horizontal text—the higher the number, the smaller the space between characters. Your choices are:

10-Pica	10 characters per inch
12-Elite	12 characters per inch
15-Fine	15 characters per inch

1 inch = 25.4mm

To make your choice, keep selecting the Print Pitch gadget until the appropriate option is displayed.

NOTE: These are the standard sizes that most printers can support for text printing. Graphic printing from desktop publishing or word processing programs can override this setting and allow you to print any size character.

Print Spacing

Spacing refers to how many lines of text are printed in 1 vertical inch of space. You can select 6 lines per inch (lpi) or 8 lpi. The higher the number, the less space there is between the lines.

1 inch = 25.4mm

If you need to change the displayed option, select the Print Spacing gadget, and the other available option will appear.

NOTE: These are the standard settings that most printers can support for text printing. Graphic printing from desktop publishing or word processing programs can override this setting.

Print Quality

This gadget determines the quality of the printout. Choosing Draft gives you a lower-quality printout but faster printing. Choosing Letter gives you a higher-quality printout but slower printing.

If you need to change the displayed option, select the Print Quality gadget, and the other option will appear.

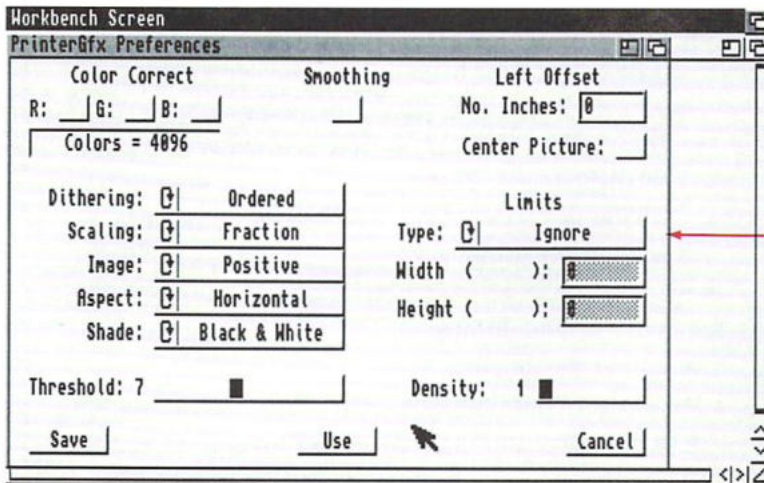
NOTE: These are the standard settings that most printers can support for text printing. Graphic printing from desktop publishing or word processing programs can override this setting.

Save/Use/Cancel

To save the settings shown in the window, select the Save gadget. If you only want to try the settings, select the Use gadget. To exit the Printer editor without making any changes, select the Cancel gadget.

The Printer Graphics Editor

Open the PrinterGfx icon and the following window appears:



PrinterGfx window

This window supports extended printer graphics features. The printer you are using should be selected through the Printer editor.

For tips on printing screen dumps, see the box on page 3-58.

Color Correct

NOTE: This can only be used with a color printer.

Color correction tries to better match the colors on your screen to the colors on the printout. You can use color correction on red, green or blue, or on a combination of the colors. To turn on color correction, select the check box next to the color, or colors, you want to correct: r (red), g (green), or b (blue).

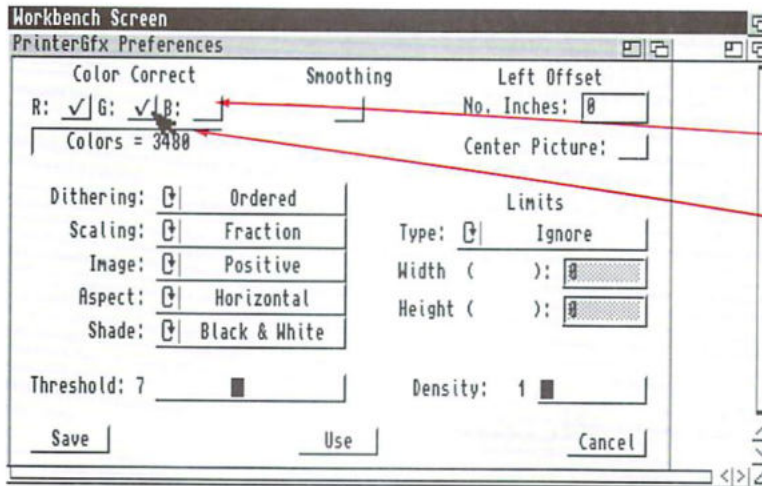
==== Tips for Printing Screen Dumps ====

For better *quality* screen dumps:

- On most printers, friction fed paper tends to produce better graphic dumps than tractor fed paper. There is less horizontal banding.
- Densities which use more than one pass should only be used for black-and-white screen dumps. If you use a multiple-pass density for a gray-scale or color dump, the output may be muddy or dark. Multiple-pass color dumps also dirty the printer ribbon (i.e., the yellow will become contaminated with other colors).

For *faster* screen dumps:

- Lower the density.
- Use horizontal dumps rather than vertical dumps.
- If you are dumping a two-color image, set Shade to black-and-white. This is much faster than a gray-scale or color dump.
- Turning on Smoothing doubles the printing time. Use Smoothing for the final copy only.
- Floyd-Steinberg dithering doubles the printing time, while Ordered and Halftone dithering cause no increase in printing time.
- If you are dumping a Hires screen that displays more than 4 colors, you can speed up the dump by moving the screen to the back of the display *once printing has started*. This is easily done by pressing left Amiga-N.



check boxes

Colors gadget

Color correction results in a reduction of the number of printed colors. When color correction is not used, all 4,096 colors displayed by the Amiga can be printed on a color printer. For each color you choose to correct, 308 shades of that color are lost. The number of colors that can be printed is shown in the Colors gadget underneath the color correction check boxes.

Smoothing

Sometimes when printing diagonal lines, the printed lines may be jagged. When **smoothing** is turned on, the Amiga attempts to smooth diagonal lines to get rid of the jagged appearance. This option is best suited for use with programs that do graphic dumps of text. When smoothing is turned on, printing may be much slower.

Floyd-Steinberg dithering cannot be used with Smoothing. If Floyd-Steinberg is selected when Smoothing is on, the dithering mode automatically changes to Ordered. (Dithering is explained later in this section.)

1 inch = 25.4mm

1/10th inch = 2.54mm

Left Offset/No. Inches

This gadget determines the number of inches to shift, or **offset**, the printed picture. This is similar to setting up a left margin. The offset is entered in increments of tenths of an inch. The Center Picture option (below) disables Left Offset.

To enter the value, select the No. Inches text gadget, delete the current value, type in the correct value, and press Return.

Center Picture

When Center Picture is turned on, the printed picture is horizontally centered on the page. To turn Center Picture on, select the check box gadget. Any value entered for the Left Offset will be ignored.

Dithering

When Shade is set to Black & White, changing the dithering method has no effect on the printout.

Just as images on your screen are made up of tiny pixels, printed images are made up of tiny dots. **Dithering** refers to the printing of dots of different colors (or shades of grey) in such a way that they are so small and close together that the eye sees them as one color. This enables you to produce printouts which appear to have more colors than the four ink colors normally available on a color printer.

For instance, where there is a pixel of black on the screen, black dots will appear on the printout. However, if you have a pixel of purple, a color printer will use dots of yellow, magenta, and cyan to create the illusion of purple. If you are printing grey scale printouts, the printer will use varying patterns of black dots to replicate the intensity of the purple on the screen.

The available dithering options are:

Ordered Color intensities are formed using an ordered pattern of dots, similar to a checkerboard pattern. The dots, while they vary in color, are of the same density and are printed in straight rows and columns. This is the standard type of dithering.

Halftone Color intensities are formed by varying the size and density of the dots. This technique is similar to the one used in newspapers and comic books. It works best on high density printers (greater than 150 dots per inch).

For instance, while a pixel of black may be reproduced with four black dots, a pixel of purple may be reproduced by two red and two blue dots of varying sizes that are placed in such a way that to the human eye they look like purple.

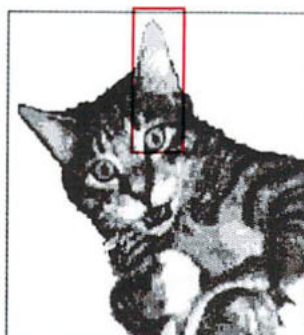
Floyd-Steinberg Color intensities are formed using the Floyd-Steinberg error-distribution method, a complex algorithmic formula. Basically Floyd-Steinberg creates a dot pattern that maximizes the image's detail by distributing the intensities of each pixel throughout the dots comprising that pixel as well as throughout the neighboring dots.

Printing may be slowed down considerably when this option is chosen.

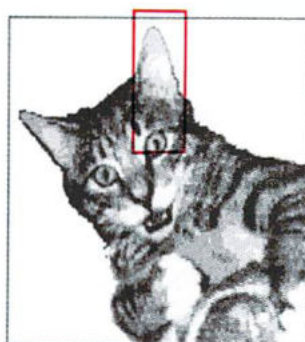
This option automatically turns off Smoothing and works best on high density printers (greater than 150 dots per inch).

For an illustration of each of the dithering methods, see page 3-62. The pictures were generated on a 300 dot-per-inch printer.

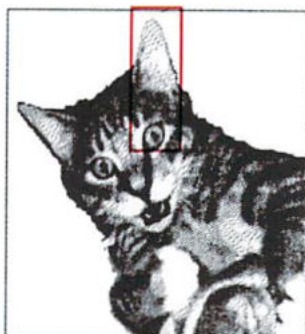
Examples of Different Dithering Output



Ordered



Halftone



Floyd-Steinberg



Scaling

Scaling refers to the process of changing the size of an image. The actual size of the printout will be determined by the Limits setting (explained later in this section). It will be scaled up or down to the nearest multiple of the width and height of the picture. The available options are:

Fraction The perspective of the picture is preserved. Pixels are enlarged or reduced at random. Select this option if you are printing pictures with lots of shading.

Integer Every pixel on the screen is guaranteed to appear as an even number of dots on the printout. Select this option when printing a picture that contains thin vertical and horizontal lines (like a grid).

For example, if the picture on the screen is 320 x 200, the printed picture will be either 320, 640 or 960 dots wide, etc., and 200, 400 or 600 dots high, etc.

Integer scaling completely overrides the Aspect setting making it possible to get a slightly distorted picture. (The Aspect setting, explained on page 3-64, determines whether the picture is printed horizontally or vertically on the page.) This option is also useful for printing out bit-image text, since the fonts will not be distorted due to fractional scaling.

This setting only affects black-and-white and grey scale printing.

Image

When set to Positive, the image is printed as it appears on the screen. When set to Negative, the image is reversed — what is black on the screen is printed as white, and what is white on the screen is printed as black. This is similar to a photographic negative.



positive



negative

Aspect

When set to Horizontal, the image is printed as it appears on the screen — what appears across the top of the screen is printed across the top of the paper. When set to Vertical, the image is printed sideways — what appears across the top of the screen is printed along the right-hand side of the paper.



horizontal



vertical

Shade

These options let you select what colors to print. Not all printers support these options. The available options are:

- | | |
|------------------|---|
| Black &
White | Colors are printed as either black or white. Whether a color is printed as black or white is determined by the threshold value (explained below). Dithering has no effect when black-and-white printing is selected. |
| Grey Scale1 | Colors are printed in varying shades of grey. |
| Color | Colors are printed as they appear on the screen. This can only be used with color printers. |
| Grey Scale2 | This option supports a maximum of four shades of grey and is used for printing pictures designed using the A2024 monitor. |

Threshold

The threshold value determines which colors on the screen are printed as white, and which are printed as black. When the setting for Image is Positive and the threshold value is low (around 2), only the darkest color on the screen is printed as black. Everything else is printed as white. Increasing the threshold value causes more colors to be printed as black.

This setting only affects black-and-white printing.

If you change the Image setting to Negative, black and white will be reversed. Therefore, a low threshold value will cause the darkest color on the screen to be printed as white.



threshold = 8



threshold = 13

Limits/Type

The Width and Height limits (explained on 3-69) allow you to specify the dimensions for your printout. However, those limits can be interpreted in several ways, dependent on the Type setting. The available options are:

Ignore

The Width and Height limits are ignored. The printed picture's size is the size requested by the application. The only restrictions are that its width cannot be greater than:

$$\frac{(\text{right margin} - \text{left margin}) + 1}{\text{characters per inch}}$$

For instance, if you are using 8.5 inch paper, with 1 inch margins, and 10 characters per inch, the width cannot be greater than 6.6 inches.

Height is restricted to the number of lines on the page divided by the lines per inch (this is usually equal to the length of the paper). For instance, if paper length is set to 66 (11 inch paper), and line per inch is set to 6, the printout cannot be longer than 11 inches.

Bounded The printed picture's size is bounded by the Width and Height limits. For example, if the printed picture should be no bigger than 4.0 x 5.0 inches (but it could be smaller), set Width to 40, Height to 50, and select Bounded. (Width and Height are interpreted in tenths of inches.)

This option is provided so that the text settings (margins, lines per page, etc.) do not have to be changed every time a graphic print is made.

Absolute The Width and Height limits are interpreted as absolute values. For example, if the printed picture should be exactly 4.0 x 5.0 inches, set Width to 40, Height to 50, and select Absolute. This completely overrides the Aspect setting (Horizontal or Vertical), making it possible to get a very distorted picture.

However, you can use Absolute to get a non-distorted printout that is a specific width or height, not both. Set either the Width or Height limit to the desired dimension, and set the other limit to zero.

For example, if Width is set to 40 and Height to 0, then the printed picture will be 4.0 inches wide and as tall as necessary in order to be in perspective. If both dimensions are set to zero, the printed picture will be as wide as possible and as tall as necessary in order to retain the picture's perspective.

Pixels

The Width and Height limits are interpreted as pixels, instead of tenths of an inch. If one of these values is set to zero, the same rules for the Absolute option apply. The printout will be the width or height specified, and as tall or as wide as necessary to retain perspective.

Multiply

The Width and Height limits are used to multiply the source picture's width and height. For instance, if you specified a Width of 2 and a Height of 4, the printed picture will be two times the source picture's width (in pixels) and four times the source picture's height. For example, if the source picture were 320 x 200 pixels, the printed picture would be 640 pixels wide and 800 pixels high.

If one of these values is set to zero, the same rules for the Absolute option apply. The picture will be scaled as necessary to maintain the proper perspective.

Width

This gadget limits the width of the printed picture. The value is interpreted as tenths of an inch unless Pixels or Multiply is the selected Type. To enter a value, select the text gadget, type the correct value, and press Return.

Height

This gadget limits the height of the printed picture. The value is interpreted as tenths of an inch unless Pixels or Multiply is the selected Type. To enter a value, select the text gadget, type the correct value, and press Return.

Density

This gadget selects the graphics print **density**. The lower the density, the faster the image will print (on those printers with multiple densities). When you select a higher density, more dots are used to create the printout and the image is sharper. However, it will take a long time for the image to print.

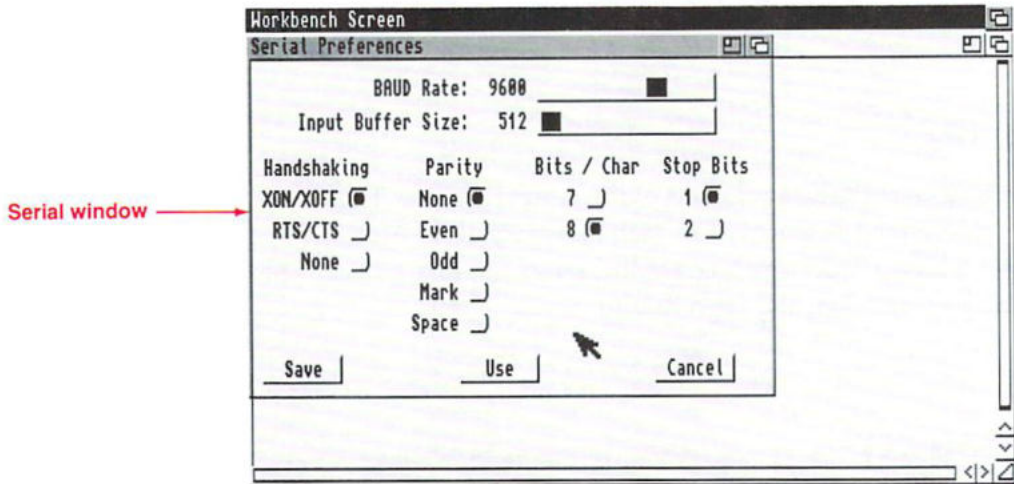
This option is not supported by every printer. You can check the specifications for your printer in Appendix B, "Printer Drivers", to determine if multiple densities are supported.

Save/Use/Cancel

To save the settings shown in the window, select the Save gadget. If you only want to try the settings, select the Use gadget. To exit the PrinterGfx editor without making any changes, select the Cancel gadget.

The Serial Editor

Open the Serial icon and the following window appears:



In order to successfully communicate through a modem or network, you must make sure that information is sent and received in a form understandable and compatible with the device with which you are communicating. This window lets you set the specifications for the **serial** port. Check the documentation packaged with your serial device to determine the appropriate settings.

Baud Rate

With serial communication, information is sent and received one bit at a time.

The **baud rate** determines the number of bits transferred through the serial port each second. Since most characters are usually 10 bits (1 start bit, 8 data bits, 1 stop bit), if you divide the baud rate by 10, you can approximate how many characters per second (cps) are transmitted.

The baud rate you select must match the rate of the device with which you are communicating. The larger the value, the faster the data is transferred. The available rates are: 110, 300, 1200, 2400, 4800, 9600, 19200, and 31250 baud. The current rate is shown to the left of the slider.

To select the correct baud rate, point to the slider bar, and drag the slider until the correct value is shown.

Input Buffer Size

The **input buffer** is an area of memory set aside for serial communication. The buffer holds incoming information that is sent to the Amiga. The available sizes are: 512, 1024, 2048, 4096, 8192, 16384, 32768, and 65536 bytes. The current size is shown to the left of the slider.

To select the buffer size, drag the slider until the desired value is shown. You may want to use a larger buffer when working with a high baud rate or when the Amiga is performing many tasks.

Handshaking

Handshaking refers to the method used to control the flow of information through the serial port and the device attached to it. The computer and the device must be set to the same handshaking method in order to communicate. The available choices are:

- XON/XOFF This is the most common method. Characters embedded in the data stream between the two devices regulate the data flow. These special characters are called XON and XOFF.

RTS/CTS	With this method, data flow is regulated via separate control lines, called RTS (Request To Send) and CTS (Clear To Send).
---------	--

NOTE: This method requires a properly wired serial cable.

None	This method causes handshaking to be shut off entirely, allowing communication between the devices without restriction or regulation. Use this option with caution.
------	---

To change the handshaking method, select the radio button next to the correct method.

Parity

Parity refers to a method for detecting transmission errors. Some computers check for errors in transmission by setting the highest bit of each character a certain way. This bit is called the parity bit. The computer checks this bit to ensure that the transmission is complete and accurate. The available choices are:

None	All bits are used for data. This should be used when Bits/Char is set to 8. No parity checking occurs.
Even	The total number of ON bits in each character should always be an even number.
Odd	The total number of ON bits in each character should always be an odd number.
Mark	The highest bit is always ON.
Space	The highest bit is always OFF.

To change the parity method, select the radio button next to the correct method.

Bits/Char

Bits per character specifies the number of bits that are sent through the serial port for each character and the number of bits expected for each character received.

Your choices are 7 or 8. This setting should correspond with your setting for parity. If parity is set to Even, Odd, Mark or Space, bits per character should be set to 7 since some systems look for parity in the 8th bit of data. If parity is set to None, you should set bits per character to 8.

To change the number of bits sent, select the other radio button.

Stop Bits

Stop bits are extra bits added at the end of a character. They allow the computer to correctly interpret spacing between words and when a transmission ends. This pertains to both characters sent and received through the serial port.

Your choices are 1 or 2. Slower-processing computers usually require two stop bits. Computers which operate at 300 baud, or faster, generally require one stop bit. If you are using 8 data bits, you can only use one stop bit, or you may lose some data during transmission.

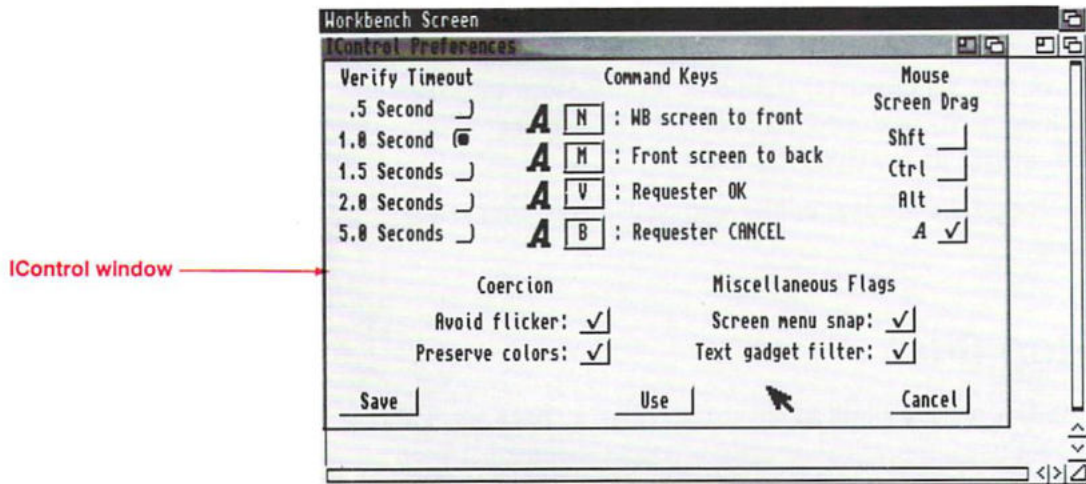
To change the number of stop bits, select the other radio button.

Save/Use/Cancel

To save the settings shown in the window, select the Save gadget. If you only want to try the settings, select the Use gadget. To exit the Serial editor without making any changes, select the Cancel gadget.

The IControl Editor

Open the IControl icon, and the following window appears:



This window lets you change several system settings, such as the default keys for moving screens or for selecting an action gadget in a requester. The different gadgets are explained below.

Verify Timeout

In rare instances, the system may be waiting for a response from a program you are running while, at the same time, the program is waiting for the system to do something.

To help prevent this kind of situation, the Verify Timeout gadget allows you to specify the amount of time that the system will wait for a response from another program. If no response is received in the allotted time, the system will proceed and avoid a deadlock.

Your choices are .5, 1.0, 1.5, 2.0, and 5.0 seconds. In general, it is best to select a longer period of time. To select a timeout setting, select the radio button next to the time of your choice. The default setting is 1 second.

Command Keys

These four gadgets let you change some of the default keys used by the Workbench. You can only specify a new letter key to be used in conjunction with left Amiga. The use of left Amiga cannot be changed. The defaults you can change are:

WB screen to front	Specifies the key used to move the Workbench screen in front of any other screens. The default is N.
Front screen to back	Specifies the key used to move the front-most screen behind all other screens. The default is M.
Requester OK	Specifies the key used to select the OK, Retry or Continue gadget shown in a system requester. The default is V.
Requester Cancel	Specifies the key used to select the Cancel gadget in a system requester. The default is B.

To enter a new key, select the text gadget, type in the new letter and press Return.

Mouse Screen Drag

Normally, you can hold down left Amiga, point anywhere in the Workbench screen, and drag the screen by holding down the selection button and moving the mouse. This gadget lets

you specify other keys that can be used in addition to or in place of left Amiga.

The available keys are left Shift, Ctrl, and left Alt. To select a key, point to the check box to its right and click the selection button.

For instance, if you select Ctrl, you must hold down Ctrl to drag the screen. If Ctrl and Shift are selected, you must hold down both Ctrl and left Shift to drag the screen.

Coercion

NOTE: These two options, Avoid flicker and Preserve colors, are only applicable when Productivity mode is selected.

When a Productivity screen is displayed, your Multiscan monitor works at a higher frequency than if an alternative mode (Hires, SuperHires, etc) is displayed. When several screens are open, the front-most screen determines the frequency of the monitor. If you drag the front screen down so that a Productivity screen and a non-Productivity screen are both visible, the back screen may be distorted since the monitor is still operating at the frequency determined by the front screen.

For instance, assume you have both a Productivity Workbench screen and a Hires paint program open. When the Productivity screen is in the front, the monitor is working at a higher frequency than when the paint program is displayed. If you drag the Productivity screen down, so that you can see both the Workbench screen and the paint program screen, the paint program screen may be distorted. This is because the monitor is still working in the higher frequency.

However, the Amiga will try to display the back screen properly, and in doing so may disturb the colors of the screen or interlace the screen. The Coercion gadgets allow you to disable

these effects. Selecting the Avoid flicker box will prevent the back screen from becoming interlaced. The Preserve colors gadget keeps the screen's original colors intact. However, selecting these options may result in an even more distorted back screen.

Screen Menu Snap

This option is provided for users who work with screens that are larger than the monitor's display area. Normally, the Workbench menus appear at the top left corner of the screen. If the left-most side of the screen is not visible, Screen menu snap shifts the Workbench screen so that the menus are still accessible. The screen only shifts while the menu button is held down.

Text Gadget Filter

This gadget controls whether control characters are recognized when entered into text gadgets. A control character is a key combination (usually Ctrl and an alphabetical key) that performs a certain function. For instance, Ctrl-M is equivalent to pressing Return.

Several control characters which perform text editing functions are listed below:

Ctrl-M	Same as pressing Return.
Ctrl-H	Deletes character to the left of the cursor (same as Backspace).
Ctrl-X	Deletes the entire line
Ctrl-U	Deletes all characters to the left of the cursor.

Ctrl-K	Deletes all characters from the cursor to the end of the line.
Ctrl-A	Moves the cursor to the beginning of the line.
Ctrl-Z	Moves the cursor to the end of the line.

When Text gadget filter is on, control characters that perform editing operations can be entered into text gadgets, and their functions will be performed. Control characters that are not recognized as having editing functions will be filtered out.

When Text gadget filter is off, control characters will be entered into the text. Special editing functions will not be available. You can insert control characters into the text gadget, whether filtering is turned on or not, by pressing Ctrl-left Amiga along with the desired alphabetical key. For instance, to enter Ctrl-M, press Ctrl-left Amiga-M.

NOTE: In certain windows with multiple text gadgets, like the IControl window, pressing Tab moves the cursor to the next text gadget. In these windows, even if Text gadget filter is off, you must press left Amiga-Tab to enter a Tab into the text gadget.

Save/Use Cancel

To save the settings shown in the window, select the Save gadget. If you only want to try the settings, select the Use gadget. To exit the IControl editor without making any changes, select the Cancel gadget.

The Workbench Configuration Editor

The Workbench Configuration Editor is no longer part of the system software. Its features have been incorporated into menu items and a new Commodity Exchange program.

To save the Workbench window as backdrop, choose the Backdrop menu item in the Workbench menu, then choose the Snapshot menu item in the Windows menu.

To select the Double-click for window to front option, use the ClickToFront Commodity program explained in Chapter 5.

The Editor Menus and Presets Drawer

Each editor has three menus: Project, Edit, and Options. These menus let you save changes to a specified file, enabling you to save different configurations of the same editor. By default these files are saved in the Presets drawer, although you can save them elsewhere if you wish. If you save icons for the files, you can implement the settings by opening the file's icon. You do not need to open the actual editor.

For instance, if you use two different printers with your Amiga, an MPS 1250 and an HP LaserJet, you can save the specifications for each printer in two different files in the Presets drawer. When you wanted to use the MPS 1250 printer, you could open the Presets drawer and just double-click on the MPS 1250 icon. The specifications would immediately take effect, although the Printer editor window would not open. If you were to open the window, it would show the settings for the MPS 1250. When you wanted to switch to the HP LaserJet, you could simply double-click on the HP LaserJet icon.

This section explains each of the menus and provides a detailed example of saving specifications for two printers.

The Project Menu

The options in this menu let you save the editor settings to a specific file. It also allows you to open previously saved files.

The options are:

- Open . . . Loads the information from a previously saved file. When you choose Open, a file requester appears.
- Save As . . . Allows you to specify the file where you want to save the currently displayed settings. A default filename in the Presets drawer is provided in the requester. If you want to use a different filename, type in the complete path, then select the OK gadget.

When you want to use those settings, open the editor, choose the Open menu item, and type in the complete path to the file. (The default path is SYS:Prefs/Presets.) Select the Use gadget, and the settings will be used until you reboot the Amiga or open another editor file.

You could also open the Presets drawer and double-click on the file's icon.
- Quit Exits the editor.

The Edit Menu

The options in this menu allow you to easily restore previously used settings or the default settings. The options are:

Reset to Defaults	Returns the editor settings to the default settings.
Last Saved	Returns the editor settings to the last saved settings.
Restore	Returns the editor to the settings that were displayed when the editor was open.

The Options Menu

This menu contains one item to allow you to save icons with your files. It is described below:

Save Icons?	Allows you to choose whether or not to save icons with the files saved with the Save As menu item. If you choose to save icons with the files, the icons will be saved in the same drawer as the file.
-------------	--

For instance, if you save printer specifications to the SYS:Prefs/Presets/Printer.pre file, the icon for the file will appear in the Presets window. If you double-click on the icon, specifications saved in the file will be implemented.

Using the Presets Drawer

Suppose you have two printers attached to your Amiga, an MPS 1250 that you use for dot-matrix printouts and an HP LaserJet that you use for high-quality, desktop publishing output. The following example shows how to save printer specifications for both printers.

1. *Open the Printer editor.*
2. *Make the appropriate selections to correspond with the HP LaserJet printer.*
3. *Choose Save As from the Project menu.*

When the requester appears, select the default file or enter a filename, such as SYS:Prefs/Presets/Laser.

4. *Without closing the editor, make the appropriate selections to correspond with the MPS 1250 printer.*
5. *Choose Save As from the Project menu.*

When the requester appears, enter a new filename, such as SYS:Prefs/Presets/MPS 1250.

6. *Select the Save gadget.*

The editor window will close and the current printer specifications will be for the MPS 1250 printer.

When you want to use the HP LaserJet:

7. *Open the Presets drawer, and double-click on the Laser icon.*

The settings for the HP LaserJet will take effect even though the Printer editor does not open.

An alternative method is to open the Printer editor, choose the Open menu item, and enter SYS:Prefs/Presets/Laser in the requester.

When you want to use the MPS 1250, open the Presets drawer and double-click on the MPS 1250 icon.

Chapter 4. The Workbench Programs

Chapter 3 taught you how to customize your Workbench and set up your Amiga to work with various peripherals. This chapter explains the rest of the drawers in the Workbench window.

Aside from Prefs, there are five other drawers in the Workbench2.0 disk window: System, Monitors, Utilities, WBStartup and Expansion. The Utilities and System drawers contain icons for programs included on the Workbench2.0 disk, such as:

- Clock, a program for displaying a clock on your screen
- AddMonitor, a tool that lets you notify the Amiga that a monitor other than the standard RGB-style monitor has been added to the system
- More, a program for displaying text files
- Say, a tool that lets the Amiga speak

The Monitors drawer contains icons used by the AddMonitor program. The WBStartup drawer allows you to automatically start different programs when you boot or power on the Amiga.

The Expansion drawer is sometimes used when you add peripherals to your Amiga. For instance, if you add a Bridgeboard to an Amiga, an icon for a specialized **library** is copied to the Expansion drawer. If a peripheral uses the Expansion drawer, it will be explained in the documentation packaged with that product.

The first thing explained in this chapter is how to use Tool Types. Tool Types let you specify different parameters for many programs. Next, the programs in the System drawer are covered, followed by those in Utilities. Finally, you'll learn how to add programs to the WBStartup drawer.

Tool Types

Before you start reading about the Workbench programs, you should take a few minutes to familiarize yourself with the concept of Tool Types. As explained in Chapter 2, Tool Types are used to specify parameters used by a program. For instance, you can use a Tool Type to change the style of clock displayed by the Clock program or to specify a file to be used by the Display program.

Tool Types are usually in the form of KEYWORD = argument. In this manual, the KEYWORD is shown in uppercase letters while the argument is in lowercase letters. However, case differences do not matter when entering the information.

You enter Tool Types in an icon's Information window. Select the appropriate icon, then choose Information from the Icons menu. When the window appears you can add, delete or change Tool Types.

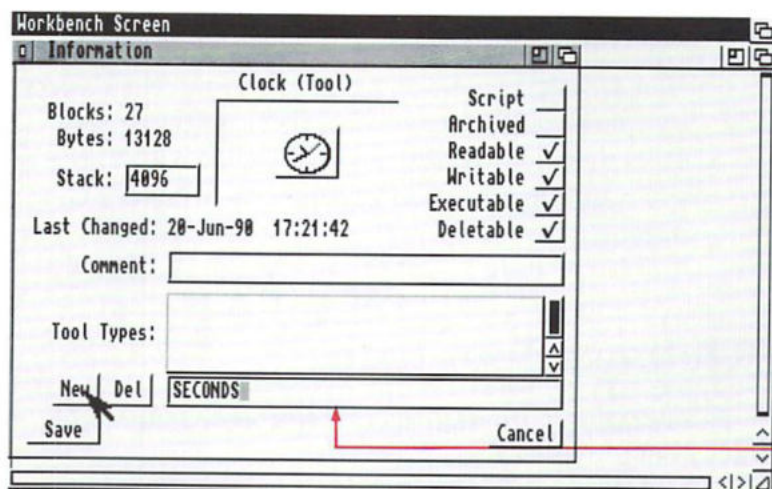
Adding a Tool Type

1. *Select the appropriate icon, then choose Information from the Icons menu.*

The Information window will appear.

2. *Select the New gadget.*

A cursor appears in the text gadget.

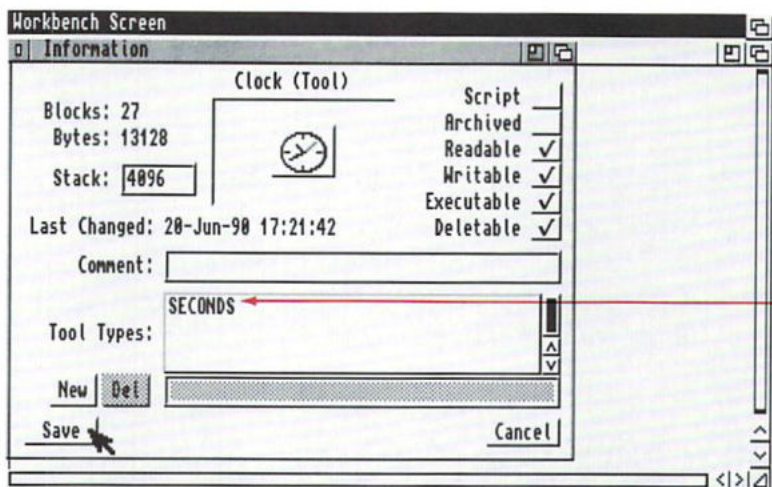


In this example, the Keyword does not take an argument.

text gadget

3. Type in the new Tool Type and press Return.

After pressing Return, the new Tool Type will appear in the Tool Types scrolling list.



new Tool Type

To add another Tool Type, repeat steps 2 and 3.

4. *Select the Save gadget to save the new information.*

If you are adding more than one Tool Type, do not select Save until you've entered all of them. Selecting Save closes the Information window. If you do not want to save your changes, select Cancel or the window's close gadget.

Deleting a Tool Type

1. *Select the appropriate icon, then choose Information from the Icons menu.*

Any Tool Types you have added will be shown in the Tool Types scrolling list.

2. *Select the Tool Type you want to delete from the list.*

Point to the Tool Type and click the selection button. The Tool Type will now appear in the text gadget.

3. *Select the Del gadget.*

To delete another Tool Type, repeat steps 2 and 3.

4. *Select the Save gadget to save the change.*

Do not select Save until you have made all of your changes. Selecting Save closes the Information window. If you do not want to save your changes, select Cancel or the window's close gadget.

Changing a Tool Type

1. *Select the appropriate icon, then choose Information from the Icons menu.*

Any Tool Types you have added will be shown in the Tool Types scrolling list.

2. *Select the Tool Type you want to change from the list.*

Point to the Tool Type and click the selection button.
The Tool Type will now appear in the text gadget.

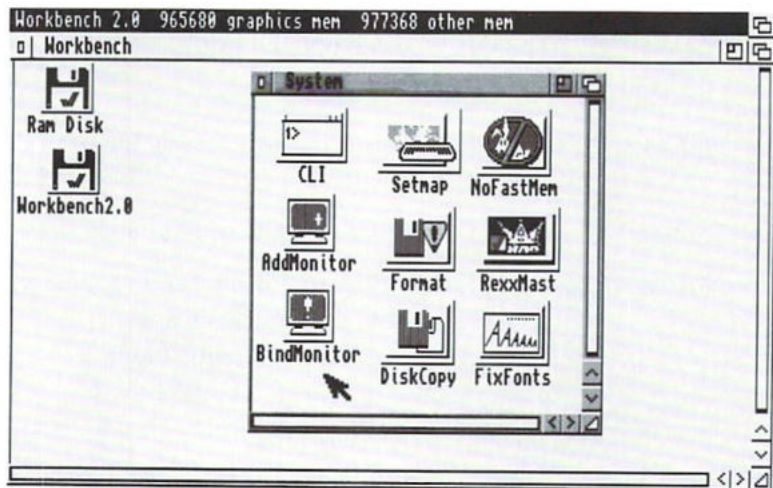
3. *Edit the text in the text gadget, making the necessary changes, then press Return.*

4. *Select the Save gadget to save the change.*

Do not select Save until you have made all of your changes. Selecting Save closes the Information window.

The System Drawer

The System drawer contains programs that have an effect on the way the system operates.



The list below explains when or why you would need to use each program. Please see the section describing the individual programs for more information.

- | | |
|-------------|---|
| AddMonitor | Is used to inform the system that you've added another type of monitor, such as an A2024 or Multiscan monitor. You can also use it if you have a monitor that supports the video standard that is not normally used in your country (PAL for NTSC countries, and vice versa). |
| BindMonitor | Assigns names (Hires, SuperHires, etc.) to the different display modes. BindMonitor is automatically run, via the Mode_Names tool, when the Workbench is started. |
| CLI | Controls the Shell, the program that lets you communicate with the Amiga through typed commands. The Shell is fully explained in Chapter 7, "Using AmigaDOS." The CLI is not explained in this Chapter. |
| DiskCopy | Is used to make copies of disks. |
| FixFonts | Is used to make new fonts accessible to the system. |
| Format | Is used to format a floppy or hard disk. |
| NoFastMem | Is used to temporarily disable any expansion RAM used by your Amiga. This is sometimes needed when using older programs that do not work properly if expansion RAM is present. |
| RexxMast | Must be running in order for programs that use the AREXX language to operate properly. A command to start RexxMast is part of the standard Startup-sequence (the file that is read when the Amiga is turned on or |

rebooted). You only need to open the REXxMast icon if you delete the REXxMast command from your Startup-sequence or if your computer doesn't have enough memory to load the program. If you're interested in learning about the AREXX language, please see Chapter 10. REXxMast is not explained in this chapter.

SetMap Tells the Amiga what keyboard you are using. (You do not need to use SetMap if you have an American keyboard.)

AddMonitor

You must use AddMonitor if you want to use the display modes available with non-RGB style monitors, such as an A2024 or Multiscan monitor. To use AddMonitor you drag an icon for the type of monitor you want to use from the MonitorStore drawer on the Extras2.0 disk to the Monitors drawer on the Workbench2.0 disk.

If you have a hard disk system, both the MonitorStore and the Monitors drawer will be in your System2.0 window.

There are icons in the MonitorStore drawer for A2024, Multiscan, PAL and NTSC monitors. The PAL and NTSC icons are for users who have monitors that support the video standard not used in their country.

For instance, if you live in Europe and use a PAL Amiga, but have a monitor that also supports NTSC displays, you must put the NTSC icon into the Monitors drawer to make the NTSC display modes available to your system. If you have an NTSC system, you do not need to put the NTSC icon into the Monitors drawer; the NTSC display modes will be your default display modes.

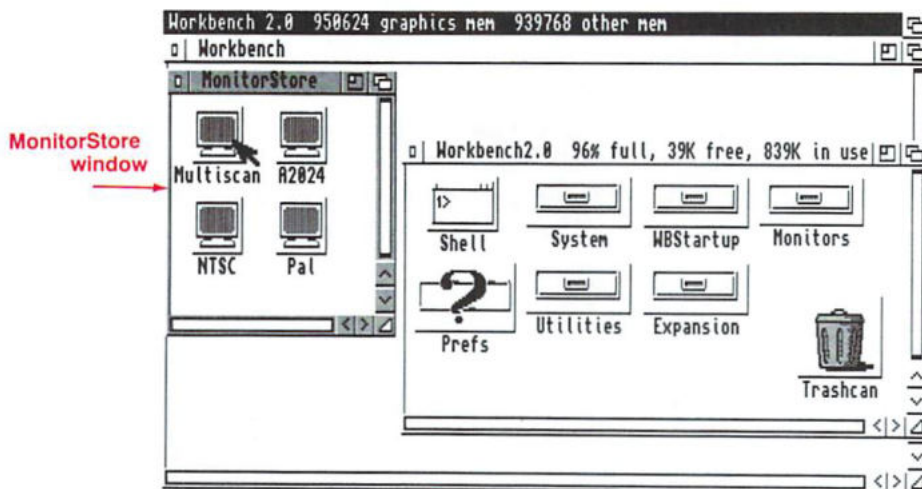


If you have a floppy disk system:

1. Open the Workbench2.0 disk window.
2. Remove the Workbench2.0 disk from the disk drive, insert the Extras2.0 disk, and open the Extras2.0 disk icon.

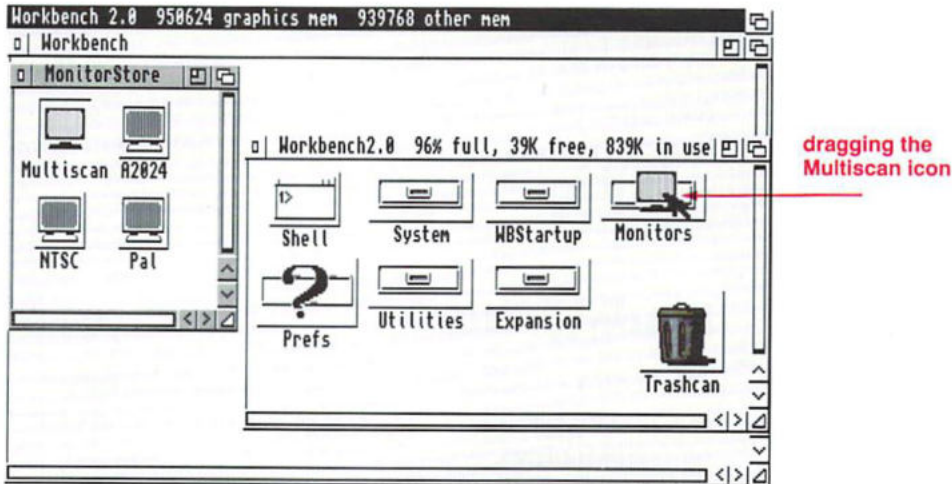
NOTE: If you have two disk drives, do not remove your Workbench2.0 disk. Just insert the Extras2.0 disk in the other drive, and open the Extras2.0 icon.

3. Open the MonitorStore drawer in the Extras2.0 disk window.



You can close the Extras2.0 disk window if you want to unclutter your screen.

4. Drag the icon for your monitor type over the Monitors drawer in the Workbench2.0 disk window.



A requester will appear asking you to insert the Workbench2.0 disk. You may have to swap between the Workbench2.0 and Extras2.0 disks until the file is completely copied.

NOTE: If you have two disk drives, you will not see this requester. The system will copy directly from one disk to the other.

If you have a hard disk:

1. Open the System2.0 disk icon.
2. Open the MonitorStore icon.
3. Drag the icon for your monitor type over the Monitors drawer.



Once the icon is copied, double-click on the monitor icon. The AddMonitor program will tell the Amiga that the monitor has been connected, and the corresponding display modes will be available through the ScreenMode editor (explained in Chapter 3.)

As long as the icon is in the Monitors drawer, AddMonitor will be run every time you boot your Amiga. You will not need to double-click on the monitor icon.



BindMonitor

The Amiga's ROM (read-only memory) contains data to support the various display modes possible with the different types of monitors. However, ROM does not automatically recognize the display modes by name. For instance, a Hires display is 640 x 200/256 pixels, but ROM does not recognize the word Hires. It only recognizes the pixel configuration of 640 x 200/256.

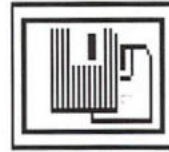
BindMonitor assigns names to the different display modes so that the system will associate the display mode with the appropriate name. To make things easier, there is a file called Mode_Names in the WBStartup window. Mode_Names already contains the assignments for all the possible display modes. As long as this icon is in your WBStartup drawer, the system will automatically be told the display mode names every time Workbench is loaded.

When you open the ScreenMode Editor in the Prefs drawer, the names assigned through BindMonitor (via the Mode_Names file) will appear in the Choose Display Mode gadget. If Mode_Names is ever removed from the WBStartup drawer, only the display modes, such as 640 x 200 or 1008 x 800, will appear in the ScreenMode window.

DiskCopy

You can copy a disk by selecting the disk icon, holding down Shift, then double-clicking on the DiskCopy icon. The copy procedure will follow the same steps as if you had chosen the Copy menu item. A requester will notify you when it is time to swap disks.

The DiskCopy program is used by the Copy menu item in the Icons menu. If you simply double-click on the DiskCopy icon, you will be referred to the Copy menu item.



FixFonts

FixFonts should be used after you have added, or deleted, information from your Fonts drawer. (The Fonts drawer does not have an icon associated with it; you have to use the Show All Files menu item in order to see the Fonts drawer on the Workbench.)

If you have a floppy disk system, the only font immediately available to you is Topaz. Topaz is stored in ROM. Other fonts are available in the Fonts drawer on the Extras2.0 disk, but to copy them to your Workbench2.0 disk, you would have to delete information from the disk to make room for them.



You should not delete files from the Workbench2.0 disk unless you are an experienced user. Instructions are given in Chapter 7, "Using AmigaDOS," for making room on the Workbench2.0 disk. If you do decide to modify your disk to make room for fonts, be sure to use a backup copy of your original Workbench2.0 disk.



If you have a hard disk system, the fonts from the Extras2.0 disk will be in your Fonts drawer on your System2.0 partition.



The Fonts drawer contains a .font file and drawer for every font available to the Workbench. For instance, inside the Fonts drawer there is a .sapphire file and a Sapphire drawer which contain the data needed to create the different sizes of the Sapphire font.

Sometimes additional font files are supplied with word processing or desktop publishing software. The software usually includes instructions on adding the files to the Fonts drawer. After you add fonts to the Fonts drawer, FixFonts updates all the .font files so that they accurately reflect the current contents of the corresponding font drawers.

To use FixFonts, double-click on the FixFonts icon. FixFonts does not open a window.



Format

You can format a disk by selecting the disk icon, holding down Shift, then double-clicking on the Format icon. The formatting procedure will follow the same steps as if you had chosen the Format Disk menu item.

The Format program is used by the Format Disk menu item in the Icons menu. If you simply double-click on the Format icon, you will be referred to the Format Disk menu item.



NoFastMem

Some very old programs may not run properly when memory other than graphics memory is present in the Amiga system, such as the expansion memory present in the A2000/A3000 or the additional memory provided by an A501 memory cartridge in the A500. In this case, double-clicking on the NoFastMem

icon forces the Amiga to use only the available graphics memory. The icon works like a toggle switch. To restore all memory to the system, double-click on the NoFastMem icon again.

SetMap

SetMap allows you to select the correct **keymap** for your keyboard. A keymap tells the computer which character to register for each key on the keyboard. The default keymap stored in ROM is *usa*—a standard American keyboard.

Additional keymaps are included on the Extras2.0 disk (in the Devs/Keymaps drawer).



Available Keymaps	
Keymap	Keyboard
cdn	French Canadian
chl	Swiss French
ch2	Swiss German
d	German
dk	Danish
e	Spanish
f	French
gb	Great Britain
i	Italian
is	Icelandic
n	Norwegian
s	Swedish
usa0	For programs developed with V1.0
usa2	Dvorak

If you have a floppy disk system, you must copy the correct keymap from the Extras2.0 disk to your Workbench2.0 disk.



If you have a hard disk system, the file will already be in the System2.0:Devs/Keymaps drawer. However, you will have to add the appropriate Tool Type to the SetMap icon's Information window. Skip ahead to the "Adding A Tool Type" section on page 4-16.

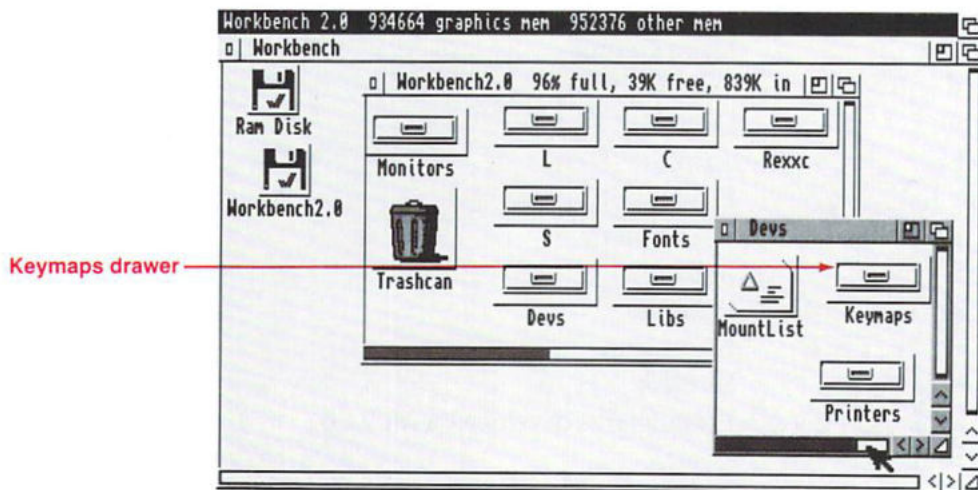
To copy a keymap to Workbench2.0:

1. *Open the Workbench2.0 disk window, and select Show All Files from the Window menu.*

Additional icons will appear in the window. Look for the icon for the Devs drawer. You may need to scroll the contents of the window, or enlarge the window, to find the icon.

2. *Open the Devs drawer.*

A window will appear containing several files and two drawers: Keymaps and Printers. (You can safely close the Workbench2.0 disk window if you want to unclutter the screen.)



3. *Remove the Workbench2.0 disk from the disk drive, insert the Extras2.0 disk and open the Extras2.0 disk icon.*

NOTE: If you have two disk drives, do not remove your Workbench2.0 disk. Just insert the Extras2.0 disk in the other drive, and open the Extras2.0 icon.

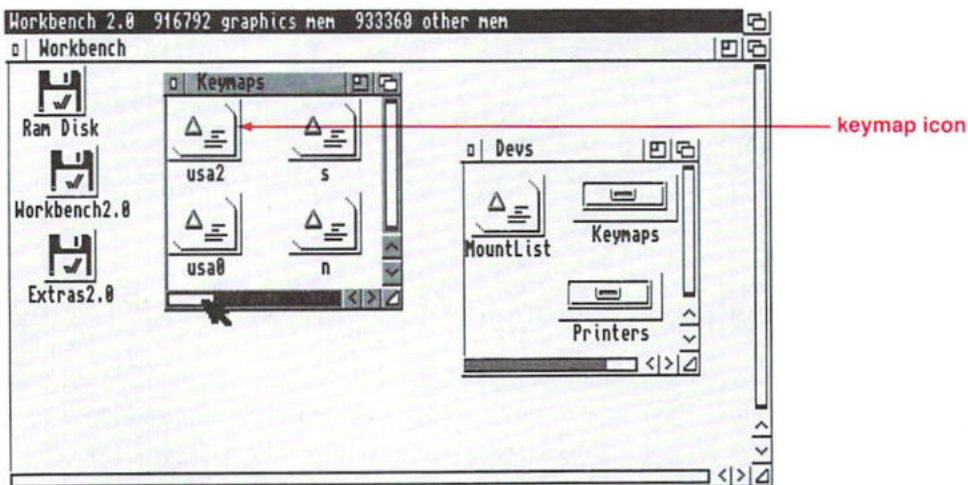
4. *Select the Extras2.0 disk window, and choose Show All Files from the Window menu.*

Just as with the Workbench2.0 disk window, an icon for the Devs drawer will appear.

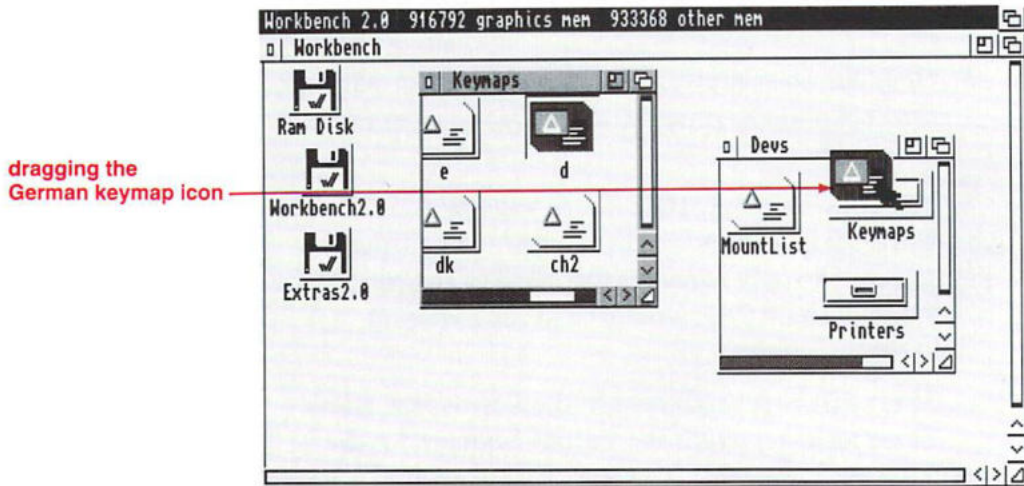
5. *Open the Devs drawer in the Extras2.0 disk window, then open the Keymaps drawer that appears in the Devs window.*

You can close the Extras2.0 disk window and the Extras2.0 Devs window if you want to unclutter the screen. The only windows that need to be open are the Workbench2.0 Devs window and the Extras2.0 Keymaps window.

There will be several icons in the Keymaps window. These icons represent the various keymaps for the different keyboards used throughout the world.



6. Find the icon that represents the correct keyboard for your use.
7. Drag the correct Keymap icon out of the Extras2.0 Keymaps window and over the Keymaps drawer in the Workbench2.0 Devs window.



A requester will appear asking you to insert the Workbench2.0 disk. You may have to swap between the Workbench2.0 and Extras2.0 disks until the file is completely copied.

NOTE: If you have two disk drives, you will not see this requester.

Adding a Tool Type

The angle brackets indicate that information, in this case a filename, must be substituted. Do not type the brackets.

Once the correct keymap is in the Keymaps drawer, you must add a Tool Type to the SetMap icon's Information window to notify SetMap of the new keymap file. The format is KEYMAP=<file>. For example, to enter the keymap for a German keyboard:

1. Select the SetMap icon, then choose Information from the Icons menu.

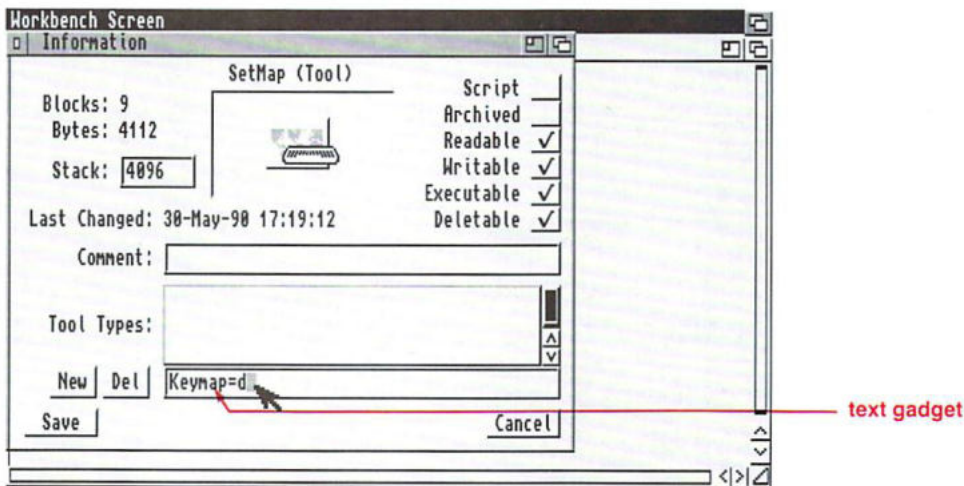
The SetMap Information window will appear on the screen.

2. Select the New gadget.

A cursor will appear in the text gadget.

3. Type:

KEYMAP = d



Press Return, and the new Tool Type will appear in the Tool Type scrolling list.

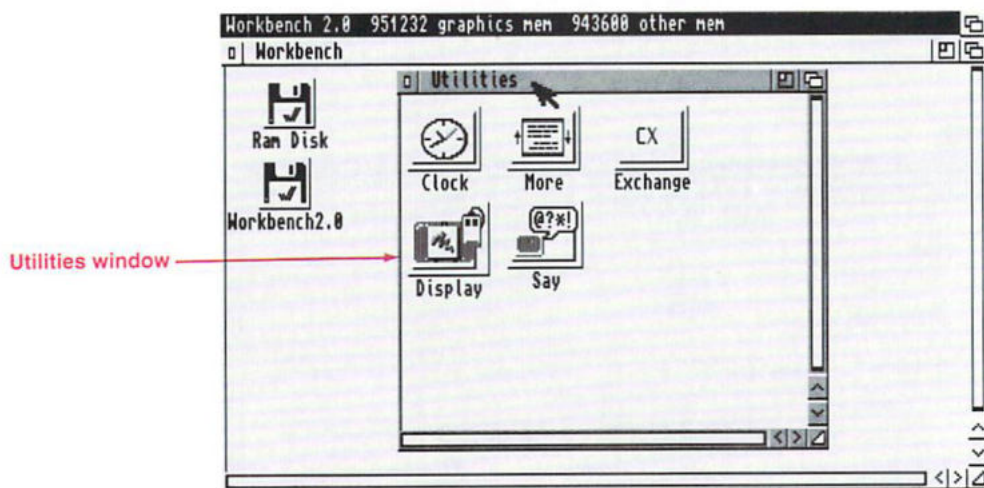
4. Select the Information window's Save gadget.

To implement your change, double-click on the SetMap icon.

You can have SetMap run automatically every time you boot your system by dragging the SetMap icon into the WBStartup drawer. This is fully explained in "The WBStartup Drawer" section on page 4-35.

The Utilities Drawer

The Utilities drawer contains some basic programs that you may want to use while working with your Amiga.



The programs are listed below:

Clock	Displays a clock on the Workbench screen.
Display	Displays an IFF graphics file.
Exchange	Monitors and controls the Commodities Exchange programs on the Extras2.0 disk.
More	Displays the contents of text files on the screen.
Say	Makes the Amiga talk.

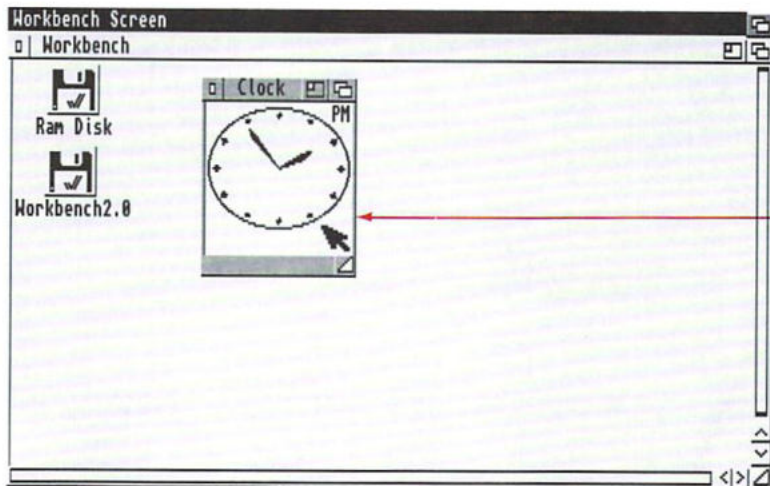
Each of these programs is explained in this section.

Clock

The Clock lets you display the time on your Workbench screen. You can also use it as an alarm clock to signal you at a specified time.



When you open the Clock icon, a window with a round clock face appears. If the time shown is incorrect, use the Time editor in the Prefs drawer to set the clock (see Chapter 3).

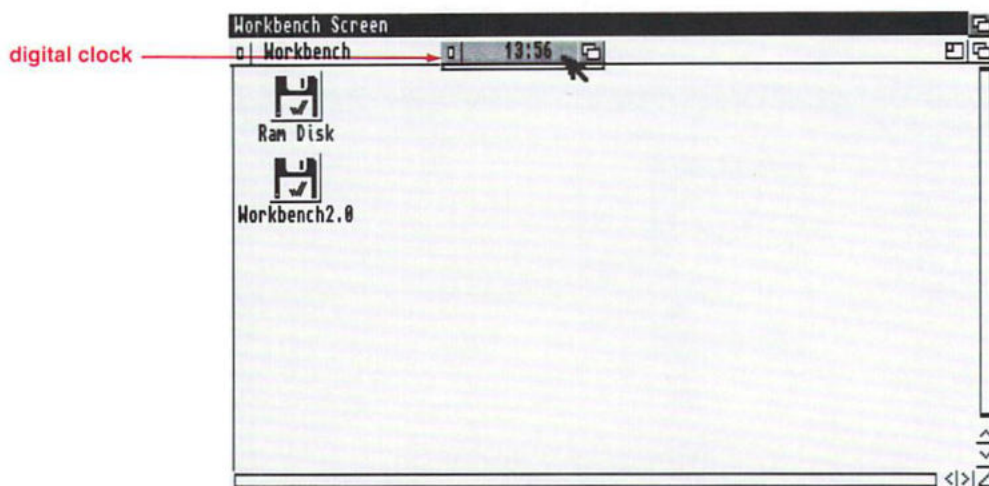


analog clock

When the Clock window is selected, menus are available that let you change to a digital display, change the way the time is shown, and set the alarm.

The Type Menu

This menu lets you change the way the clock is displayed. If you choose Analog, the round clock appears. (This is the default.) This is the only mode that lets you change the size of the Clock window. If you choose Digital, a digital clock the height of the title bar appears.



The Mode Menu

This menu lets you choose a 12-hour or 24-hour clock. If you choose the 12 Hour menu item, AM or PM is shown after the time. If you choose 24 Hour and a digital display, time is displayed as 0:00 (12:00 AM or midnight) through 23:59 (11:59 PM). 12 Hour is the default.

The Seconds Menu

This menu lets you choose to display the seconds. If you choose the Seconds On menu item, a second hand appears on the analog display. In a digital display, the seconds are shown after the minutes (11:36:04). Seconds Off is the default.

The Date Menu

This menu lets you choose to display the date. If you choose the Date On menu item, the date is shown underneath the analog clock. If you are using the digital clock, the date and time will be alternately displayed. Date Off is the default.

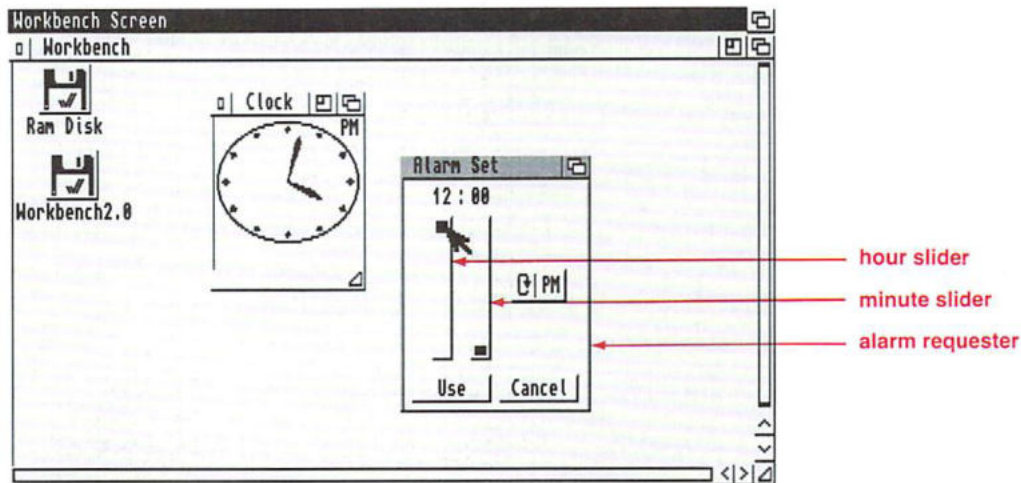
The Alarm Menu

With this menu, you can tell the Amiga to signal you at a certain time. The signal is a brief flash on the display. If your monitor is hooked up to the Amiga's audio output, you will also hear a short tone.

To set the alarm:

1. *Choose the Set menu item.*

In the requester that appears, time is shown in the same mode as your clock — either 12 Hour or 24 Hour. If it is shown in 12 Hour mode, an AM/PM cycle gadget will appear next to the time sliders.



The requester defaults to 12:00. It does not reflect the currently set time.

2. *To change the setting for the hour, use the hour slider.*

Point to the bar in the hour slider, and press the selection button. Use the mouse to drag the bar up or down until the correct hour is displayed at the top of the slider.

3. *To change the setting for the minutes, use the minute slider.*

As with the hour slider, drag the slider bar until the correct minute is displayed.

4. *When the requester displays the desired alarm time, select the Use gadget.*

If you want to restore the previous alarm setting, select the Cancel gadget.

5. *To turn the alarm on, choose the Alarm On menu item.*

When the clock reaches the alarm setting, the screen will flash and a short tone will sound.

The alarm will remain on and will flash at the same time each day, until you choose Alarm Off. If the Clock is not displayed, the alarm will not work. The next time you open the Clock, you will have to reset the alarm.

Tool Types

By entering Tool Types in the Clock icon's Information window, you can save the menu, size and position settings so that the Clock will open as you want it every time. The acceptable Tool Types are:

- | | |
|---------|--------------------------------------|
| DIGITAL | The clock will open in digital mode. |
| 24HOUR | The clock will open in 24-hour mode. |

SECONDS	The clock will display the seconds.
DATE	The clock will display the date.
LEFT = <n>	The clock will open <n> pixels from the left edge of the screen.
TOP = <n>	The clock will open <n> pixels from the top of the screen.
WIDTH = <n>	The clock will be <n> pixels wide (disregarded if using a digital clock).
HEIGHT = <n>	The clock will be <n> pixels high (disregarded if using a digital clock).

Display

Display allows you to view graphic files saved using the standard Amiga **IFF ILBM** format. Most graphics created with standard Amiga paint programs save files in this format. You can create a slide show with Display by specifying multiple files to be shown. You can advance from one picture to the next automatically, or you can tell Display to wait for a mouse click.

To display a single picture file:

1. *Select the Display icon.*
2. *Hold down Shift, and double-click on the icon of the picture file.*

The file will be displayed on a new screen.



To return to the Workbench screen, press Ctrl-C. You can also click on the hidden close gadget in the upper left corner of the screen.

An alternative method is to add a Default Tool to the picture icon's Information window. If you do this, you will be able to display the picture by opening its icon.

1. *Select the picture icon, then choose Information from the Icons menu.*

An Information window will appear.

2. *Select the Default Tool text gadget.*

A cursor will appear in the gadget. Delete any existing text.

3. *Type in the path to the Display program as the picture's default tool.*

Be sure to specify the complete path, such as Workbench2.0:Utilities/Display.

If you have a hard disk system, the complete path is System2.0:Utilities/Display.

4. *Select the Save gadget.*

If you open the picture icon, the Display program will be run, and the picture will be displayed.

To display multiple files:

1. *Select the Display icon.*
2. *Hold down Shift, and select the icons for the picture files you want to display.*
3. *When you get to the last picture file, double-click on its icon.*

The first picture file will be displayed. Press Ctrl-C to advance to the next picture. To exit Display before all the pictures are shown, press Ctrl-D.



You can also create an **ASCII** text file containing a list of the IFF files that you want to display. This “filelist” can be created with one of the Workbench text editors, ED or MEMacs, or with any word processor that allows you to save files in ASCII format. For instance, a sample file might look like this:

```
PicsDisk:Art/Sea  
PicsDisk:Art/Mountain  
PicsDisk:Art/Sky  
PicsDisk:Art/Lightning  
PicsDisk:Art/Gulls
```

ASCII is a standard text format that can be read by many programs and computers.

This indicates that the IFF files are in the Art drawer on a disk called PicsDisk. The filelist should be saved in a drawer accessible by the Workbench, and it should have a project type icon. (You can check this by opening the icon’s Information window. If it is not a project, you can use the IconEdit program to change the icon’s type. See Chapter 5 for instructions on using IconEdit.)

Open the filelist icon’s Information window, and add the following Tool Type:

```
FILELIST = true
```

Select the Default Tool text gadget and enter the path to the Display program, for instance, Workbench2.0:Utilities/Display. This tells the system that the file is a project of the Display Tool. When you open the filelist icon, Display will automatically be run, and the pictures in the list will be displayed.

Tool Types

Display supports several Tool Types for moving from one picture to the next. You can move forward by clicking the selection button or by specifying a certain amount of time for Display to wait before moving ahead. Several of these Tool Types are global, which means they must be added to the Information window for the filelist icon or for the Display icon.

The global Tool Types are listed below:

FILELIST = true	Display all the pictures listed in the ASCII file.
MOUSE = true	Clicking the selection button displays the next picture file. Clicking the menu button displays the previously displayed picture.
LOOP = true	When Display has shown all of the selected picture files, it will loop back, or start over, with the first picture. This will continue until you press Ctrl-D.
BACK = true	The picture files will be displayed on a back screen. The Workbench screen will remain active and at the front of the display. This is useful when you are printing picture files while doing something else.
PRINT = true	Display will automatically print each file that is displayed. You can also print manually by pressing Ctrl-P while the picture is displayed on the screen.
AUTOSCROLL = true	If the screen is larger than the display area, it will scroll automatically when the pointer is moved to the edge of the screen.
TIMER = <seconds>	Display will automatically advance from one picture to the next in the specified number of seconds. For instance, if you enter TIMER = 5, Display will automatically move ahead to the next picture after 5 seconds have passed.

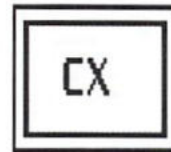
The angle brackets indicate that information, in this case a number, must be substituted. Do not type the brackets.

The following Tool Types can be entered on a per-picture basis.

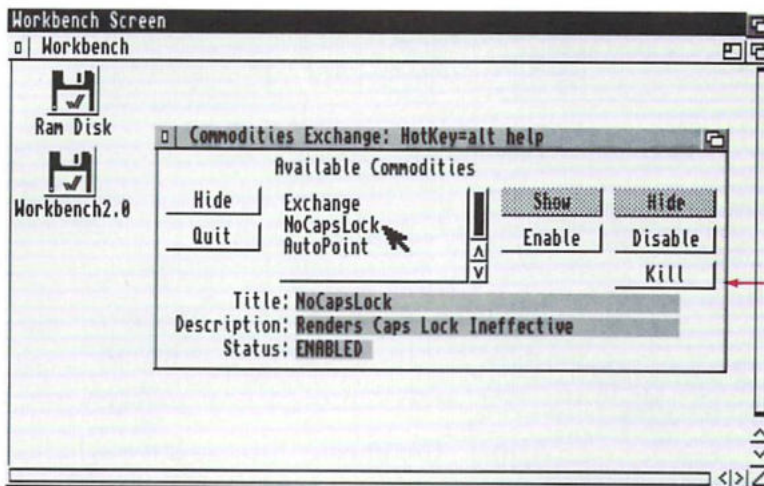
EHB = true	Unspecified, 6-bitplane pictures are treated as Extra Halfbrite rather than HAM.
NOTRANSB = true	Borders around the picture are not transparent when genlocked.
VIDEO = true	The picture file will be displayed with a full-video display clip.

Exchange

Exchange lets you monitor and control the Commodities Exchange programs stored in the Tools/Commodities drawer on the Extras2.0 disk. (For more information on the individual Commodities programs, see "The Commodities Drawer" section of Chapter 5.)



When you open the Exchange icon, a window appears.



Exchange window

Any Commodities that have been opened will be displayed in the Available Commodities scroll gadget. When you select a Commodity from the scrolling list, information about that program appears beneath the gadget:

Title	Shows the name of the selected Commodity.
Description	Gives a brief description of the program.
Status	Shows whether the program is enabled or disabled.

The gadgets to the right of the scroll window control the selected program. Exchange also has an Action menu that contains menu items corresponding to each of these gadgets. You can perform the same operation by choosing the menu item or using the menu item's keyboard shortcut.

The operations are explained below:

Show	Brings the window for the selected Commodity to the front of the screen. If the window is closed, Show automatically opens it. If the selected Commodity does not open a window, this gadget will be ghosted.
Hide	Closes the window for the selected Commodity <i>but does not exit the program</i> . This is the same as selecting the window's close gadget or choosing the Hide menu item from a Commodity's menu. If the selected Commodity does not open a window, this gadget will be ghosted.
Disable	Temporarily turns off a Commodity.
Enable	Turns a Commodity back on if it has been disabled.

- Kill** Exits the currently selected Commodity program. If the program has a window, selecting the Kill gadget is the same as choosing the Quit menu item.

The gadgets to the left of the Available Commodities scroll gadget control the Exchange window. There are corresponding menu items in Exchange's Project menu. These operations are explained below:

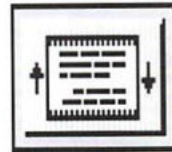
- Hide** Closes the Exchange window, but the program continues to run.
- Quit** Shuts down Exchange so that it is no longer monitoring the other Commodities.

More

More allows you to display ASCII text files on the Workbench screen.

To run More, select the More icon, then while holding down Shift, double-click on the text file icon. If the text file does not have an icon, use the Show All Files menu item in the Window menu to display a pseudo-icon for the text file.

You can also run More without specifying the text file by double-clicking on the More icon. In this case, a file requester will ask you for the complete path to the text file you want to read. This file requester contains a pattern gadget that allows you to view all the files that match the specified pattern. Pattern matching is fully explained in Chapter 7, "Using AmigaDOS."



A typical More display looks like this:

```

:/$/startup-sequence
version >NIL:
Failat 21
SetClock >NIL: load
resident >NIL: c:List pure add
resident >NIL: c:Copy pure add
resident >NIL: c:Assign pure add
copy >NIL: ENVARC: ram:env all quiet noreq
mkdir 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
--- More (66%) ---

```

The message at the bottom of the window, --- More (66%) ---, indicates the percentage of the file viewed so far.

To move through the display, use the following key sequences:

Space bar	Displays the next page.
Backspace	Displays the previous page.
Return	Displays the next line.
<	Displays the first page.
>	Displays the last page.
%n	Displays approximately n% into the file; if you type %60, you will be moved 60% into the file.
Ctrl-L	Refreshes the window.
/<text>	More will perform a case-sensitive search for the text specified after the slash (/).
.<text>	More will perform a case-insensitive search for the text specified after the period (.).

The angle brackets indicate that information must be substituted. Do not type the brackets.

N	Finds the next occurrence of the previously searched for text.
H	Help (displays a list similar to this one).
Q	Exits the program.
Ctrl-C	Exits the program.
Shift-E	Allows you to edit the file using the editor set in ENV:editor (see Chapter 7 for more information on environment variables).

A case-sensitive search means that More will look for the text exactly as it is entered. If you type the text in capital letters, More will only look for occurrences of the text that appear in capital letters.

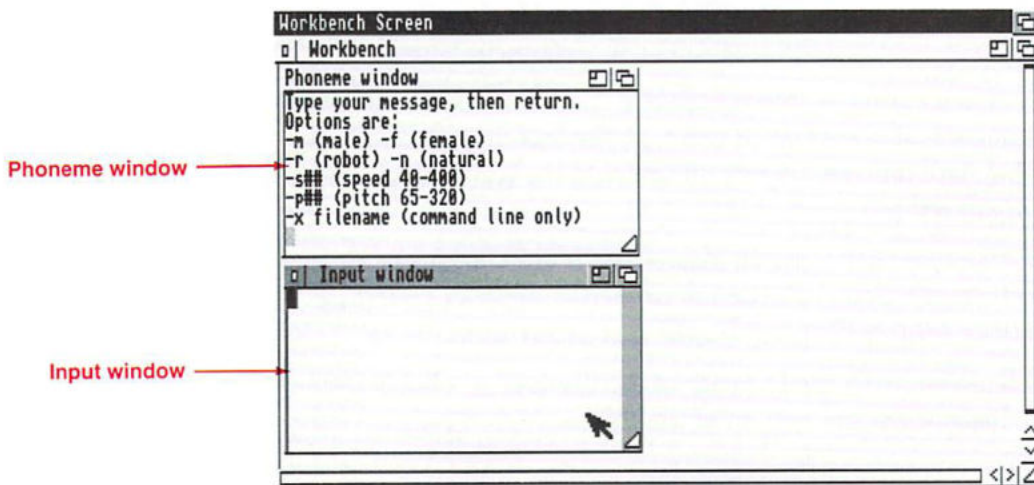
A case-insensitive search means that it does not make a difference whether the text is entered in upper or lowercase letters. More will search for the text in any form.

When you reach the last page of the display, --- End of File --- is displayed at the bottom of the screen.



Say

With Say, the Amiga can speak words typed on the screen. When you open the Say icon, two windows appear.



The top window is the Phoneme window, and the bottom window is the Input window. You enter text in the Input window, and the text is displayed phonetically in the Phoneme window and spoken through the Amiga's stereo output connectors.

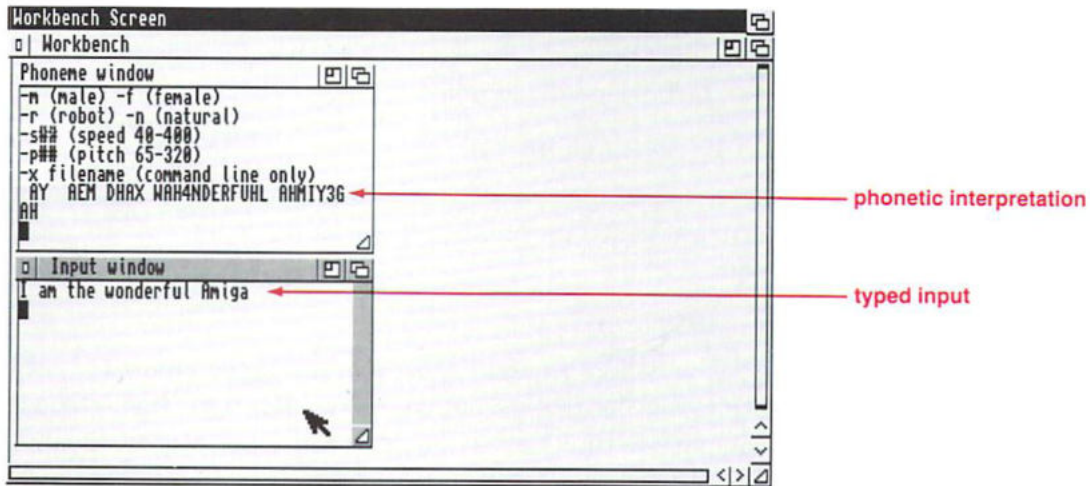
To use Say:

1. *Select the Input window, then type a word, phrase, or sentence.*

If your text reaches the edge of the Input window, do not press Return. The text automatically wraps around to the next line.

2. *After you've entered your text, press Return.*

The Amiga will literally say the sentence, while at the same time the text will appear in the Phoneme window. This is the phonetic interpretation of your input.



In some cases, it may help to spell a word phonetically (as it sounds) to get the Amiga to repeat it correctly.

You can change the voice, pitch and speed of your Amiga's speech by choosing any of the different options shown in the Phoneme window.

1. *Select the Input window.*
2. *Type the letter, or letters, necessary to make your changes.*

Your choices for voice and inflection are:

- | | |
|-----------------|-----------------------|
| -m male voice | -r robot inflection |
| -f female voice | -n natural inflection |

When you change the voice, you can also change the pitch so the change in the new voice is noticeable.

Say only understands English phonetics. To have the Amiga pronounce other languages, the input words must be spelled according to English pronunciation rules. For instance, the German phrase "die Weite" should be spelled "dee Vytah" for more correct pronunciation.

Type `-p` followed by a number from 65 through 320—the higher the number, the higher the voice's pitch. Do not put a space between the `p` and the number.

To change the speed of the voice, type `-s` followed by a number from 40 through 400—the higher the number, the faster the voice speaks. Do not put a space between the `s` and the number.

3. *Press Return.*

For example, to use a deep male voice with a natural inflection that speaks at a moderate pace, select the Input window and type:

```
-mn -s125 -p65
```

Then press Return. Remember that the hyphen must be typed before the alphabetical option. Next, enter some text. When you press Return, the Say program will speak your text using the new options.

To exit the Say program, select the Input window's close gadget. You can also select the Input window and press Return without entering any text. Either of these actions will close both the Input and Phoneme windows.

Tool Types

Say supports Tool Types for all the voice, pitch and speed options. Instead of entering the options in the Input window, you can enter them in the Say icon's Information window. This saves your selections so that they do not have to be re-entered each time you open Say. The Tool Types are:

-m	Male voice
-f	Female voice
-r	Robot inflection
-n	Natural inflection

- p# Sets the pitch; # represents a number ranging from 65 through 320.
- s# Sets the speed; # represents a number ranging from 40 through 400.

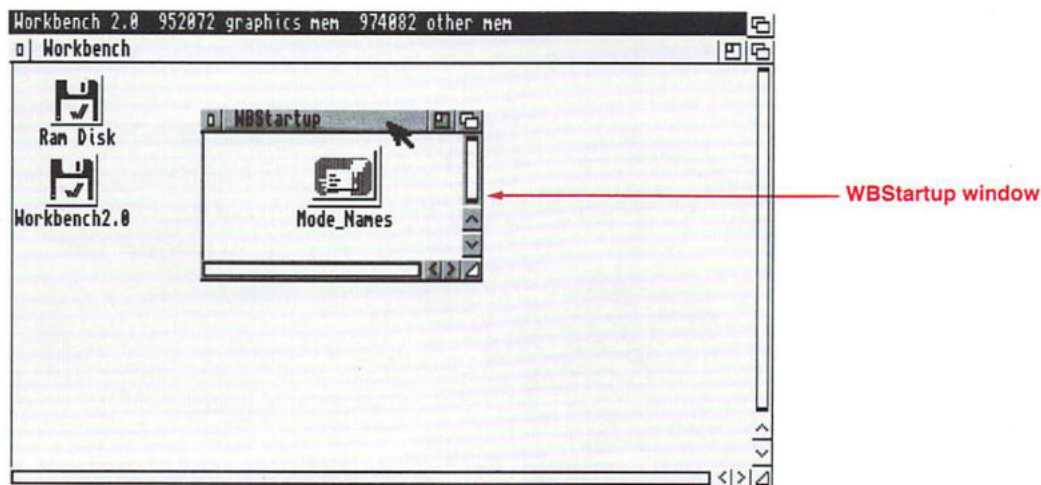
When you run Say, the options entered as Tool Types will be used, unless you specify other options in the Input window.

The WBStartup Drawer

Any icons stored in the WBStartup window will be opened whenever the Workbench is started. For instance, if you drag the Clock icon into the WBStartup window, the Clock program will be run when you reboot, or power on, your Amiga.

If you are using a non-American keymap, you may want to put the SetMap icon into the WBStartup drawer. This way the correct keymap will be used each time you boot your Amiga.

The Mode_Names icon will already be in the WBStartup window.



Tool Types

The icons that are put into the WBStartup window support a few special Tool Types:

DONOTWAIT

Normally the Workbench waits for one program to finish executing before it opens the next program. If you specify the DONOTWAIT Tool Type, the Workbench will open all the programs at once. DONOTWAIT does not take an argument.

If you do not specify the DONOTWAIT Tool Type, a requester may appear and state that the program has not yet returned. The system will ask you if it should wait some more. Select the No gadget in the requester to continue.

WAIT = <seconds>

Lets you specify how many seconds the Workbench should wait before opening the next icon in the WBStartup window.

STARTPRI = <priority>

Lets you assign a priority to an icon so that it opens before, or after, other icons. By default, all icons have a priority of 0. The acceptable range is from -128 to +127—the higher the value, the higher the program's priority.

The angle brackets indicate that information, in this case a number, must be substituted. Do not type the brackets.

Chapter 5. The Extras Programs

In Chapter 4, you learned about all the programs supplied on the Workbench2.0 disk. This chapter explains the programs found on the Extras2.0 disk, such as:

- Colors, which lets you change the colors of a screen
- GraphicDump, which prints entire screen images
- IconEdit, which lets you create and change icons
- KeyShow, which displays the keymap for your Amiga

If you have a hard disk, the Extras programs will be in the Tools and MonitorStore drawers in your System2.0 window.

When you are finished with this chapter, you should be comfortable with the Workbench system and with running programs through the Workbench. Hard disk users should continue on and read Chapter 6, "Using a Hard Disk."

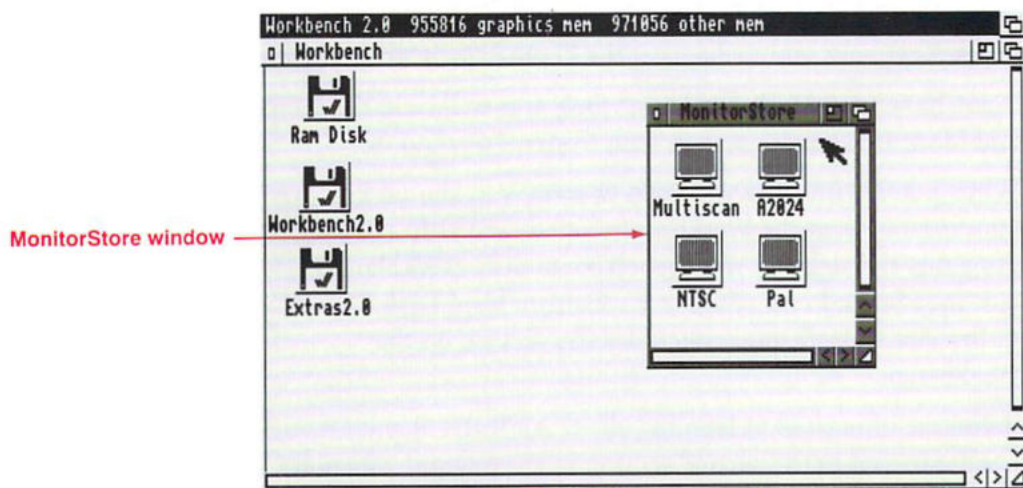
Otherwise, this chapter concludes the section on the Workbench.

If you are interested in learning about AmigaDOS and the Shell, proceed to the second section of the manual. Chapters 7 through 9 explain how to use AmigaDOS and the Amiga editors.



The MonitorStore Drawer

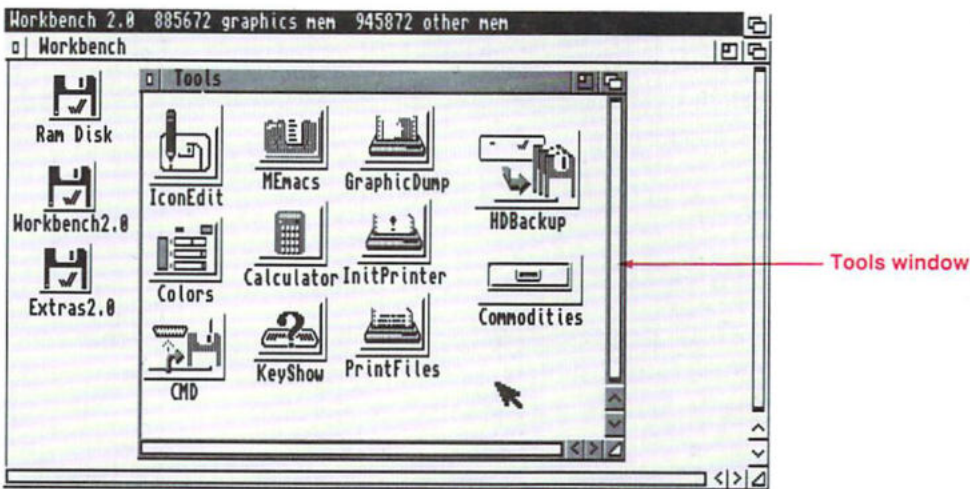
This drawer contains icons for A2024, Multiscan, PAL and NTSC monitors. These icons represent projects used by the AddMonitor program (explained in Chapter 4) and provide an easy way for you to notify your system that you have attached one of these monitors.



See the "AddMonitor" section of Chapter 4 for full instructions on adding an A2024, Multiscan, PAL or NTSC monitor to your system.

The Tools Drawer

The Tools drawer contains programs that expand your printing options, allow you to change the color of non-Workbench screens, even create new icons.



The programs are listed below:

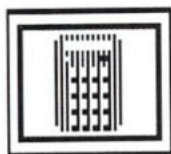
Calculator	A standard, four-function calculator.
CMD	Allows you to redirect printer output to a file.
Colors	Changes the colors of a non-Workbench screen.
GraphicDump	Allows you to print screen images.
HDBackup	A hard disk backup and restore program, described in Chapter 6.
IconEdit	Allows you to change and create icons.
InitPrinter	Initializes your printer.

KeyShow	Displays the current keymap.
MEmacs	A text editor, described in Chapter 9.
PrintFiles	Sends files to the printer.

The programs in the Commodities drawer are explained starting on page 5-27.

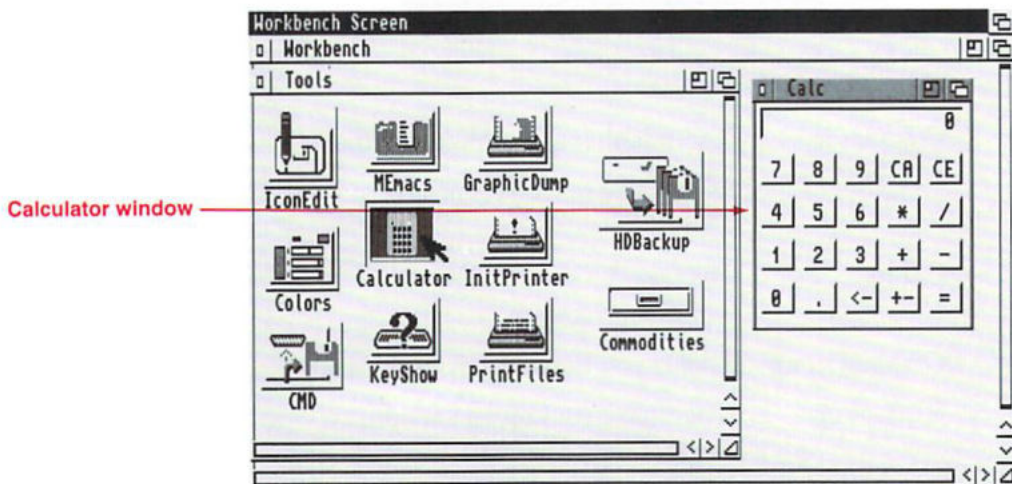


If you have a hard disk system, there may be additional icons in your Tools window that are not shown here, such as HDToolbox, Park, and Drive Definitions. These programs pertain to hard disk specific operations and are fully explained in Chapter 6.



Calculator

The Calculator is a standard four-function calculator that you can use to add, subtract, multiply and divide. Open the Calculator icon, and the calculator appears.



It works like any standard calculator — you enter numbers and use the operations keys to reach an answer. The “buttons” on the calculator are gadgets. The numbered gadgets represent the digits 0 through 9. The non-numerical gadgets represent:

- CA Clear all previous entries. Resets the calculator to 0.
- CE Clear the current entry. If you make a mistake typing, select this gadget, and re-enter your value.
- Multiply
- / Divide
- + Add
- Subtract
- . Decimal point
- <– Delete the last digit entered
- + – Change the sign of the current entry. Positive numbers become negative; negative numbers become positive.
- = Display the result of the operation.

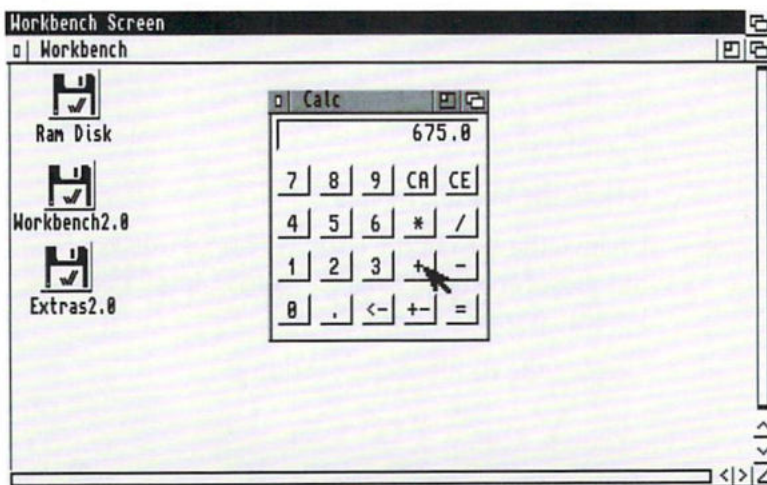
To press a button, select the gadget with the mouse or press the corresponding key on the keyboard. The <- sign corresponds to Backspace.

For instance, to add 675 and 916:

1. Select the gadgets for 675 — 6, 7, and 5.

As you select each gadget, the corresponding number will be displayed in the calculator window.

2. Select the + gadget.



3. Select the gadgets for 916 — 9, 1, and 6.

4. Select the = gadget.

The sum, 1591, will be displayed in the window.

To do the same calculation using the keyboard, you could just type 675, press +, type 916, then press Return. As you typed each number, it would be displayed in the calculator window.

To exit the Calculator, select the close gadget.

CMD

CMD directs your printer output to a file. This is useful if you do not have a printer attached to your Amiga. You can capture your output on disk, then take the disk to another Amiga that is connected to a printer. You can also use it to save the images from GraphicDump (explained later in this chapter).

Before you can use CMD, you must tell the program some details about how your system is set up and where you want the printer output to be sent. This is done by adding Tool Types to the CMD icon's Information window.

As shipped, CMD is set to certain default Tool Types. If your choice is the default, you do not need to enter anything in the Tool Types window. The recognized KEYWORDS and arguments are:

- DEVICE = <port> The Amiga port where your printer is attached, either parallel or serial.
DEVICE = parallel is the default.
- FILE = <filename> This is the name of the file where you want the printer output to be sent.
FILE = ram:CMD_file is the default.
- SKIP = true This tells CMD to skip any short initial write. Sometimes, especially with screen dumps, the first write sent to the printer is a printer reset. You can use SKIP = true to ignore that first write.
- The default is SKIP = false—initial writes are not skipped.
- MULTIPLE = true This tells CMD to intercept more than one file.
- The default is MULTIPLE = false—only one file is redirected.



The angle brackets indicate that information must be substituted. Do not type the brackets.

NOTIFY = true

This tells CMD to display progress messages.

When CMD intercepts the file, a typical message that may appear is:

Redirected <# of bytes> from
parallel.device to <filename>

After the output is sent to the file and CMD is turned off, another message may state:

CMD redirection of parallel.device removed

The default is NOTIFY = false—
messages are not displayed.

To use CMD, double-click on its icon. The next time you send information to your printer, it will be sent to the designated file instead.



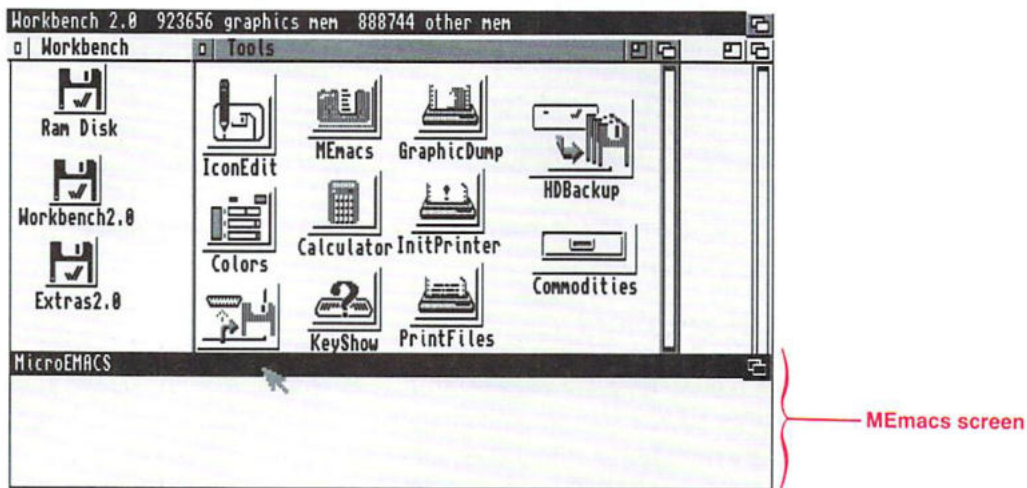
Colors

Colors lets you change the colors of a non-Workbench screen, such as a screen opened by an application program like a communications program or word processor. However, color changes made with Colors are only temporary. They cannot be saved to disk.

To change the colors of a screen, Colors must open on that screen. The following example will show you how to open Colors on the MEMacs screen. (Don't worry about using MEMacs at this point. It is fully explained in Chapter 9, "Editors.")

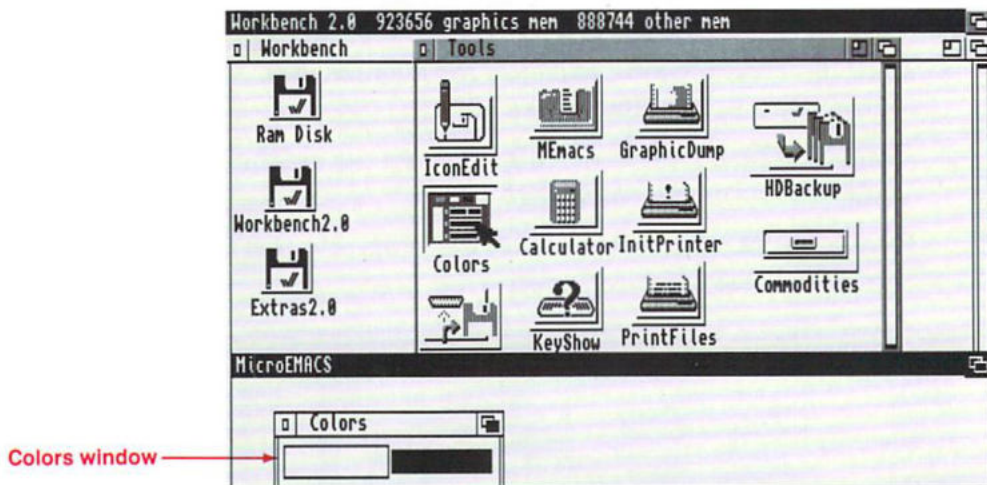
1. *Open the Tools window, and open the MEMacs icon.*
MEMacs opens on another screen.
2. *Drag the MEMacs screen down so that you can see the Tools window on the Workbench screen.*

Point to the MEMacs title bar, hold down the selection button, and drag the screen down.



3. Open the Colors icon in the Tools window.

The Colors window will open on the front-most screen which will be the MEmacs screen.

**4. Drag the MEmacs screen back up to the top of the display.**

Using Colors

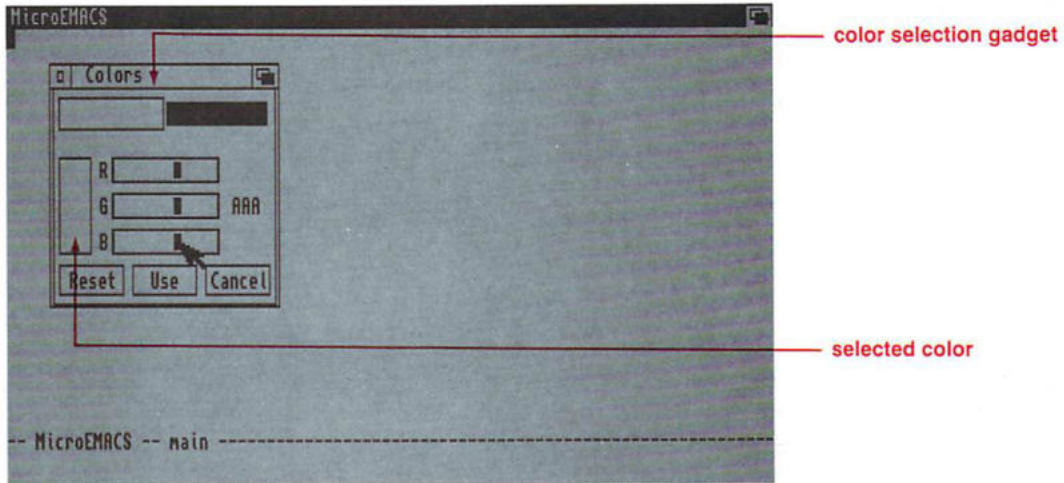
Just like the Prefs Palette editor, the Colors window contains color sliders that let you change the application screen colors. The number of colors is equal to the number of colors in the screen. In this example, the MEmacs screen is only made up of two colors, so Colors will only have two colors in its color selection gadget.

1. Select the color you want to change from the color selection gadget.

The selected color appears in the display box that runs along the left side of the Colors window.

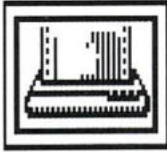
2. *Use the color sliders to change the selected color.*

Drag the slider bars in the R (red), G (green), and B (blue) color sliders until you create the color you want.



To return to the original screen colors, select the Reset gadget. To implement your changes, select the Use gadget. Select the Cancel gadget to exit Colors without making any changes.

To exit MEmacs, click on the MEmacs screen and press Ctrl-C.



GraphicDump

GraphicDump prints, or dumps, entire screens, including menus and icons, just as they appear on your monitor. Your printer must be capable of printing graphic images. Most dot-matrix printers can print GraphicDump output.

Before using GraphicDump, make sure the settings in the Printer and PrinterGfx editors are appropriate for your printer. You can specify the dimensions of the printout with the Limits setting in the PrinterGfx editor. Otherwise, the printout will be the full width allowed by the printer.

To use GraphicDump, double-click on its icon. In about ten seconds the front-most screen image will be sent to the printer. The ten second delay gives you time to open or close windows, display menus, or move screens.

Tool Types

GraphicDump supports a SIZE Tool Type. The acceptable arguments for SIZE and the resulting size of the printout are:

SIZE = tiny	1/4 the total width allowed by the printer.
SIZE = small	1/2 the total width allowed by the printer.
SIZE = medium	3/4 the total width allowed by the printer.
SIZE = large	Full width allowed by the printer. (default)

The height of the printout is such that the perspective of the screen is maintained. The Limits Type gadget in the PrinterGfx editor must be set to Ignore for GraphicDump to recognize these arguments. Otherwise, the size of the printout is determined by the Limits setting.

To specify specific dimensions in a Tool Type, use:

SIZE = <xdots>:<ydots>

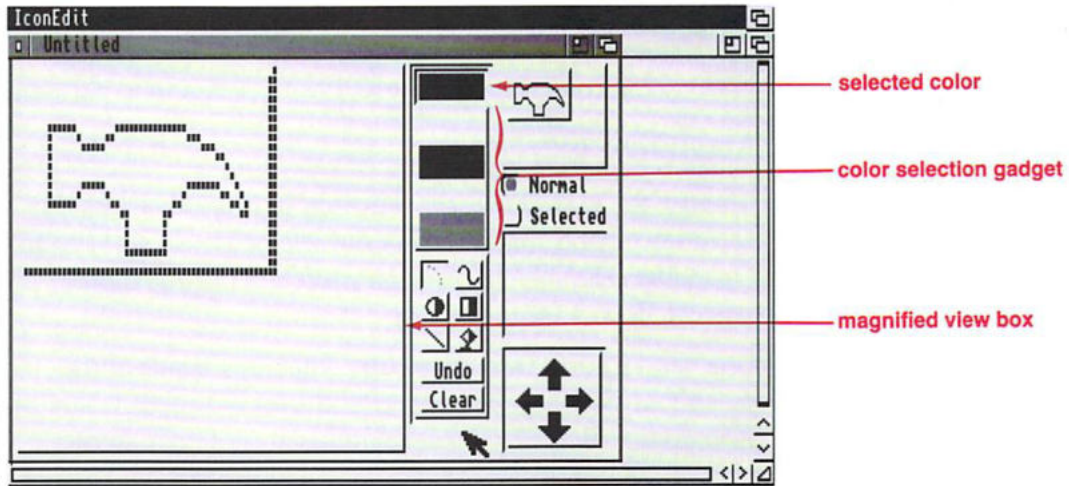
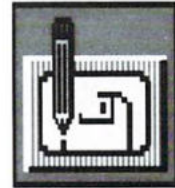
Substitute the width, in number of dots, for the <xdots> argument and the height for the <ydots> argument.

Do not include the brackets around the numbers. They just indicate that a substitution must be made.

IconEdit

IconEdit lets you personalize your Workbench by changing the appearance of existing icons and creating new icons.

Open the IconEdit icon, and the following window appears:



Just as with the WBPatten and Pointer editors in the Prefs drawer, you can draw and edit an icon in the magnified view box. However, the IconEdit window also contains several gadgets to give you more control over your drawing, such as gadgets for drawing squares, circles, and straight lines.

Color Selection Gadget

This gadget lets you select a color for drawing. In addition to the standard way of selecting a color by pointing to it and clicking the selection button, this color selection gadget also allows you to choose two colors at once.

To create a “checkerboard” pattern, select the first color, hold down Shift, then select the second color. If you were to create a solid box or circle, it would be filled with dots of both colors.

To create a pattern of vertical bars, select the first color, hold down Alt, then select the second color.

Keyboard shortcut: Press P to cycle forward through the colors. Press Shift-P to cycle through dithered patterns using the background color. Press / to reset the color to a solid pattern.

Magnified View Box

You use the mouse to draw your icon in this box. Click the selection button, and a pixel of the selected color will appear. Hold down the selection button while moving the mouse, and you can draw with the mouse.

The pointer turns into cross hairs when it is inside of the magnified view box. The center of the cross hairs is where the new pixel will appear. Pixel coordinates are shown in the IconEdit window title bar to help you with the placement of your images.

Freehand Gadget



This gadget allows you to draw unstructured shapes very quickly. If you select this gadget, then draw in the magnified view box, the pixels will quickly fill in as the mouse passes over them. However, you may not get a continuous line, and some pixels may not be filled in. This gadget is useful when you just want to sketch an icon, then go back and fill in the details later.

Keyboard shortcut: Press S to select the freehand gadget.

Continuous Freehand Gadget



This gadget is very similar to the freehand gadget except that you will always get a continuous line. However, in order to achieve a continuous line, you cannot draw as fast as you can when using the freehand gadget. The pixels will not be continually filled in as you move the mouse. There may be a delay before the display catches up with your movement.

Keyboard shortcut: Press D to select the continuous freehand gadget.

Circle Gadget



To draw a circle:

1. *Select the circle gadget.*
2. *Point inside the magnified box at the point where you want the center of the circle, hold down the selection button, and move the mouse.*

As you move the mouse, you'll draw a circle.

3. *Release the selection button when the circle is the size that you want.*

Selecting the right portion of the gadget lets you draw a filled circle. You can fill the circle with a dithered pattern, by selecting two colors from the selection gadget (as explained on page 5-14).

Selecting the left portion of the gadget lets you draw an outline of a circle. To double the thickness of the circle's outline, hold down Ctrl before releasing the selection button (step 3 above).

Keyboard shortcut: Press E for an outlined circle; Shift-E for a filled circle.

Box Gadget



To draw a box:

1. *Select the box gadget.*
2. *Point inside the magnified box at the point where you want a corner of the box, hold down the selection button, and move the mouse.*

As you move the mouse, you'll draw a box.

3. *Release the selection button when the box is the size that you want.*

Selecting the right portion of the gadget lets you draw a filled box. You can fill the box with a dithered pattern by selecting two colors from the color selection gadget (as explained on page 5-14).

Selecting the left portion of the gadget lets you draw an outline of a box. To double the width of the box's outline, hold down Ctrl before releasing the selection button (step 3 above).

You can automatically draw a three-dimensional box like the type that surrounds the Workbench icons by holding down an Alt key while drawing a box outline. To draw an "unselected" box hold down left Alt. To draw a selected box, hold down right Alt.

Keyboard shortcut: Press R for an outlined box; Shift-R for a filled box.

Line Gadget



To draw a straight line, select the line gadget, then point to the place where you want the line to start. Hold down the selection button, move the mouse to where you want the line to end, then release the selection button.

As with the circle and box gadgets, you can double the thickness of the line by pressing Ctrl before releasing the selection button.

Keyboard shortcut: Press L to select the line gadget.

Fill Gadget



You can use the fill gadget to fill an area of the magnified view box with the selected color. This is an easy way to change the color of a complete area. Assume there is an icon with text in it, and you want to change the color of that text. Simply select the new color, select the fill gadget, then move the pointer inside one of the letters and click the selection button. The letter will change to the new color.

NOTE: The fill gadget will not work on a dithered pattern.

Keyboard shortcut: Press F to select the fill gadget.

Undo

Undo

Select Undo to cancel the last mouse action that took place in the magnified view box.

Keyboard shortcut: Press U to select the Undo gadget.

Clear

Clear

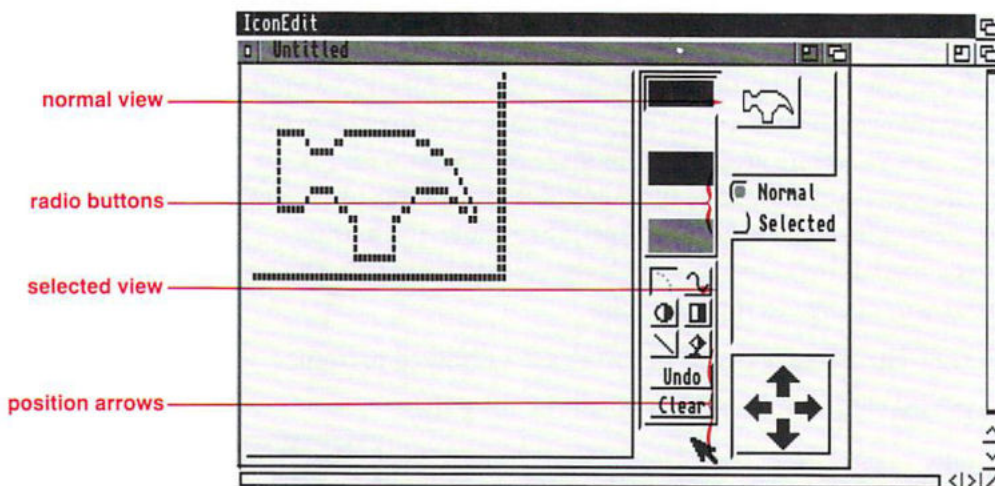
Select the Clear gadget to erase the contents of the magnified view box. The magnified view box will fill with the currently selected color.

Keyboard shortcut: Press Shift-C to select the Clear gadget.

Normal/Selected Radio Buttons

The Normal and Selected radio buttons let you switch between unselected and selected images for an icon. The normal image is how the icon will look when it is unselected. The selected image is how the icon will appear when you click on it.

When the Normal radio button is selected, any image drawn in the magnified view box will appear in the normal view box at the top of the window.



When the Selected radio button is selected, you can create the image that will appear when the icon is selected. You can only select this radio button when the Image menu item is chosen from the Highlight menu. (All the menus are explained later in this section.) Any image you create will appear in the selected view box.

Keyboard shortcut: Press Shift-S to select the Selected radio button; Shift-N for the Normal radio button.

Arrows

The arrows let you shift your image. By pointing to an arrow, and holding down the selection button, you can move the image in the magnified view in the direction of the arrow. You can use these arrows to control the placement of your image within the box surrounding the finished icon.

Keyboard shortcut: Press the corresponding cursor key to move the image.

The Project Menu

The items in the Project menu let you open and save icon files.

New AN

Loads the default icon for the currently chosen type of icon.

(The type of icon is determined by the Type menu, see page 5-20) If you have made any changes to the window that have not been saved, a requester will ask you if you want to save those changes.

Open . . . AO

Opens an existing icon file. A requester appears so that you can enter the name of the file that you want to open.

Save AS

Saves an existing icon file, overwriting the file. For instance, if you opened a file called TestIcon, then you made changes to that icon, choosing Save would save those changes to the TestIcon file. The previous contents will be lost.

Save As . . . AA

Assigns a filename to the current image. A requester lets you enter the name of the file where you want the image to be saved.

Save As Default Icon AD

Saves the current image as the default icon used when you choose the Show All Files menu item in the Workbench Window menu or the New menu item in the IconEdit Project menu.

For instance, if you create a drawer icon, then choose Save As Default Icon, that icon will be used to represent drawers when you choose the Show All Files menu item.

Quit AQ

Exits the IconEdit program. If you have not saved the current image, a requester will ask if you want to save the image before exiting IconEdit.

The Edit Menu

The items in the Edit menu allow you to import IFF ILBM clips that were created with other paint packages. To do this IconEdit uses the **clipboard**, an area of memory that is used to store text and graphics while they are being transferred between programs.

Cut **A X**

Deletes the image in the magnified view box and copies it to the clipboard.

Copy **A C**

Copies the image in the magnified view box to the clipboard.

Paste **A V**

Copies any image in the clipboard to the magnified view box, replacing the current contents.

Open Clip...

Copies an existing IFF file into the clipboard. A requester appears so that you can enter the name of the file you want to open.

Save Clip As...

Saves the current contents of the clipboard to a specified file.

Show Clip

Displays the contents of the clipboard using the Display program. If the Display program cannot be found, for instance, if the Workbench2.0 disk is not in a drive, Show Clip will not work.

The Type Menu

The items in the Type menu allow you to specify the type of icon you are changing or creating.

Disk **A 1**

Represents the disk icons that appear in the Workbench window.

Drawer **A 2**

Represents the drawer icons that appear in a disk window, such as the Utilities or Tools drawer.

Tool **A 3**

Represents a tool, such as the Calculator, Clock or IconEdit program.

Project **A 4**

Represents a project, an icon that has been created by a tool, such as the Mode_Names icon or any of the icons in the MonitorStore drawer.

Garbage **A 5**

Represents the Trashcan drawer.

The Highlight Menu

The items in the Highlight menu allow you to determine how an icon will appear when it is selected.

Complement **A 7**

Highlights the entire icon, including the background of the box surrounding the icon. For instance, if you are using the default Workbench colors and the icon is surrounded by a field of grey, the grey will become blue when the icon is selected.



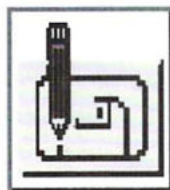
unselected



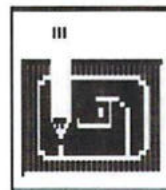
selected

Backfill**A8**

Highlights the icon, but not the background of the box. For instance, if you are using the default Workbench colors and the icon is surrounded by a field of grey, the grey will remain grey when the icon is selected.



unselected



selected

Image**A9**

Creates an entirely different image for the selected icon (an animated icon). For instance, the drawer icons on the Workbench are animated. When you select a drawer, it does not change color. Instead an entirely new image of an open drawer appears.

The Images Menu

The items in the Images menu let you manipulate the images in the normal and selected view boxes and import IFF images created with other graphic programs.

Exchange**AE**

Swaps the images that appear in the normal view and the selected view.

Copy**AC**

Copy is dependent on which radio button is selected. If Normal is selected, the image in the normal view is copied to the selected view. If the Selected radio button is selected, the image in the selected view is copied to the normal view.

Use Template

Copies a box the same size as the standard Workbench icon box into the magnified view. You can then create your new icon within this box.

Load

Loads previously saved images. When you point to the Load menu item, a submenu appears. The available submenu items are:

Normal Image **AY**

Loads the unselected image of the specified icon into the normal or selected view box, depending on which radio button is selected.

Selected Image **AU**

Loads the selected image of the specified icon into the normal or selected view box, depending on which radio button is selected.

Both Images **AI**

Loads both the normal and selected images of the specified icon into the appropriate view boxes.

IFF Brush **AJ**

Allows you to load an IFF file created by another program as either the normal or selected view, depending on which radio button is selected.

When you choose an item from the submenu, a requester appears to allow you to specify the file that you want to load. You must specify the correct drawer and filename.

Save IFF Brush **AK**

Saves an image as an IFF file.

Restore **AR**

Returns the IconEdit window to the state it was in when you opened the window or last selected New or Open.

The Extras Menu

The items in the Extras menu control a few miscellaneous features of IconEdit.

Recolor AM

Switches the colors of any pixels using the second and third colors in the color selection gadget. By default the second color is black and the third color is white. If you are using the default colors, choosing Recolor will make all white pixels black, and vice versa.

Auto TopLeft AT

Moves the image to the upper left corner of the magnified view box.

Color Palette...

Opens the Preferences Palette editor so that you can change the default colors.

The Settings Menu

The items in the Setting menu allow you to save various IconEdit options.

Use Grid

When Grid is chosen, each pixel in the magnified view box is distinct. You can see the background color surrounding each pixel. A check mark next to the menu item indicates that this option is turned on. When Use Grid is not chosen, the pixels blend together smoothly. The default is for the grid to be on.

Save Icons?

If Save Icons is chosen and you save the contents of the magnified view box with the Save IFF Brush menu item, an icon will be saved with the IFF file. If Save Icons is not chosen, no icon will be saved. The default is for icons to be saved.

Save Settings

Saves all of the current IconEdit settings, including the size and position of the IconEdit window, the size and position of the file requesters, and all of the menu item settings.

Tool Types

IconEdit supports the following Tool Types:

UNIT = <n>	Specifies the clipboard unit to use. The default is 0.
XMAG = <n>	Allows you to enlarge the width of the magnified view box. XMAG accepts a number from 4 to 16. The default is 4.
YMAG = <n>	Allows you to enlarge the height of the magnified view box. YMAG accepts a number from 4 to 16. The default is 4.
LEFTEDGE = <n>	Specifies where to place the left edge of the editor window.
TOPEDGE = <n>	Specifies where to place the top edge of the editor window.
FRLEFTEDGE = <n>	Specifies where to place the left edge of the file requester, relative to the editor window. For instance, FRLEFTEDGE = 0 will align the left edge of the file requester with the left edge of the editor window.
FRTOPEDGE = <n>	Specifies where to place the top edge of the file requester, relative to the editor window.
FRWIDTH = <n>	Specifies the width of the file requester.
FRHEIGHT = <n>	Specifies the height of the file requester.

PALETTE = <path>	Specifies the complete path to the Palette editor. This is used when the Color Palette menu item is chosen. The default is SYS:Prefs/Palette. You will only need to change this if you have moved your Palette editor.
SHOWCLIP = <path>	Specifies the complete path to the utility used to display the clipboard. The default is SYS:Utilities/Display. If you only have one floppy drive, you may want to copy the Display program onto your Extras2.0 disk, and change the path to reflect this. You could also change this Tool Type if you have another program you would rather use for displaying the clipboard.
NOICONS	Disables the creation of icons when saving support files, such as when saving a file as an IFF brush.
NOGRID	Disables the use of the grid in the magnified view box.
ICONDRAWER = <path>	Specifies the default drawer to be used by the file requesters that appear when the Open and Save As menu items in the Project menu are chosen.
ILBMDRAWER = <path>	Specifies the default drawer to be used by the file requesters that appear when the Load and Save IFF Brush menu items in the Images menu are chosen.

CLIPDRAWER = <path>	Specifies the default drawer to be used by the file requesters that appear when the Open Clip and Save As Clip menu items in the Edit menu are chosen.
ALTDRAWER = <path>	Specifies the default drawer to be used by the file requesters that appear when the Load menu item is chosen from the Images menu.
SRC	Creates a Save As C . . . menu item in the Project menu. This allows you to save the icon as C source code.



InitPrinter

In Chapter 3, you learned how to use the Printer and PrinterGfx editors to specify your print options. InitPrinter sends the printer options to the printer. This is known as **initializing** your printer.

To use InitPrinter:

1. *Turn on your printer.*
2. *Double-click on the InitPrinter icon.*

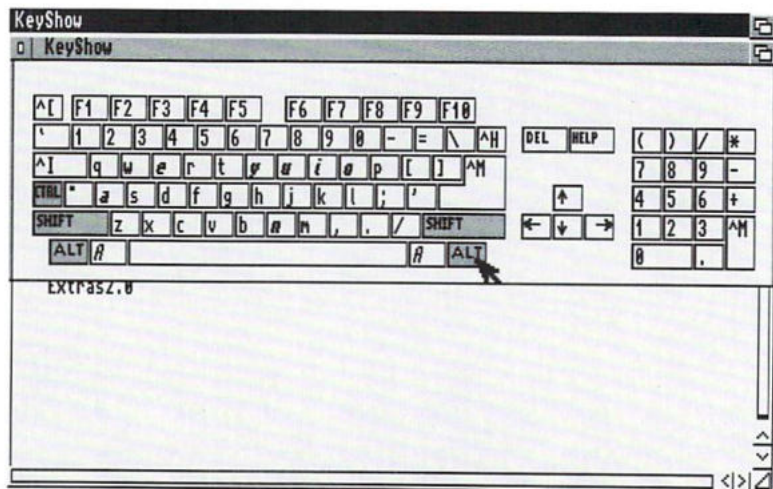
You may hear your printer reset. This is normal. It just means that the printer is receiving the information from the Amiga and is processing it.



KeyShow

The KeyShow program allows you to view the current keymap for your model of Amiga. When you open the KeyShow icon, the following window appears:

This display shows an A3000 keyboard with a USA keymap. If you are using another Amiga model or keymap, your display may vary.



The initial display shows the characters that appear when a key is pressed alone. For instance, the Q key shows a lower case q. However, when you press a **qualifier** key with a character key, you may get different output. (For KeyShow the acceptable qualifier keys are Ctrl, both Shift keys, and both Alt keys.)

To see the characters that are output when a qualifier key is pressed simultaneously with a character key:

1. *Select any of the qualifier keys that appear in the KeyShow window.*

The qualifier key will be highlighted to represent it being pressed. The KeyShow display will change to indicate the output that you get if you press the selected qualifier key along with a character key. You can select any combination of qualifiers and the display will change accordingly. Select the qualifier key again to return it to its unpressed state.

Keyboard shortcut: Instead of pointing to the qualifier key in the display, you can simply press the corresponding key on the keyboard.

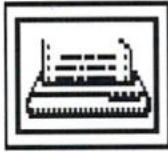
The following list is a guide to interpreting the KeyShow display:

- Grey keys are qualifier keys not currently pressed. For example, when you first open the KeyShow window, Ctrl, Shift, and Alt appear in grey. This is because KeyShow is not using those keys in the initial display.
- Blue keys are **dead keys**. A dead key is one which modifies the output of the key pressed immediately afterward. For instance, on the American keyboard, the Alt-G combination is a dead key representing the grave accent. If you press Alt-G, then press E, you will superimpose the accent symbol over the e (è).

These colors correspond to the default colors used by the Workbench.

NOTE: This multiple-key stroke procedure does not apply to Del, Help, the functions keys, or the cursor keys.

- Bold-italics indicate that a key may be used in conjunction with a dead key. In the above example, E can be modified by a dead key.
- \$\$ indicates that it would take more than one character to define the key.
- If a character is preceded by a tilde (~) or a caret (^), it is a control character.
- Blank keys are undefined for the currently selected qualifier(s).



PrintFiles

PrintFiles sends files to your printer. PrintFiles accepts multiple files, so you can use drag selection or extended selection to specify a series of files to be printed. If PrintFiles cannot find or open one of the files, it will skip it and go on to the next one.

To use PrintFiles:

1. *Select the icon of the first file you want to print, hold down Shift and select the icons of any additional files you want to print.*

You can also use drag selection to select the icons.

2. *Hold down Shift, and double-click on the PrintFiles icon.*

When printing multiple files, you may want to add a form feed between each file. A form feed starts each file on a new page. Without a form feed, the next file will start printing immediately after the first file ends.

For instance, if your first file stops in the middle of the page, the second file will start printing on that same piece of paper. If you add a form feed, the second file will begin on a new sheet.

Tool Types

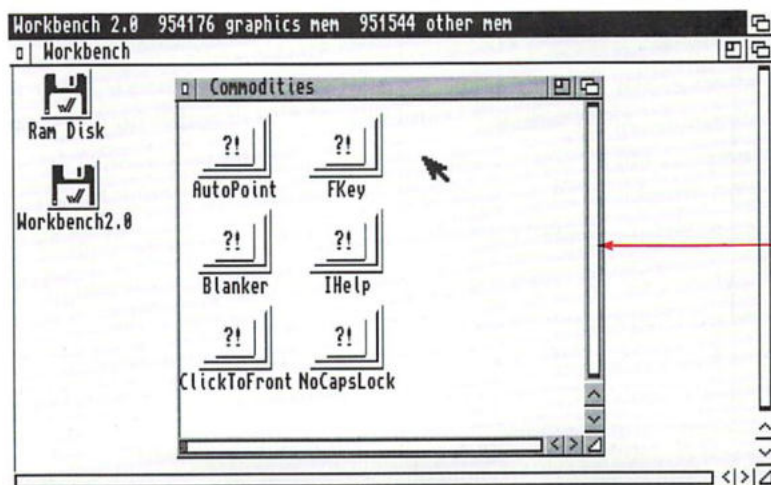
To add form feeds between files and at the end of a file, add a FLAGS=formfeed Tool Type to the PrintFiles Information window.

The Commodities Drawer

The Commodities drawer is in the Tools drawer and contains the Commodities Exchange programs. These programs monitor your keyboard and mouse input to the Amiga before Workbench or any other application programs, such as a paint program or communications program.

The Exchange program in the Utilities drawer on the Workbench2.0 disk monitors and controls all the other Commodities programs, which are shown below:

Exchange is explained on page 4-27.



Commodities window

AutoPoint	Automatically activates the window under the pointer.
Blanker	Causes the screen to go blank if there has been no input for a specified period of time.
ClickToFront	Allows you to bring a window to the front of the screen by double-clicking in it.
FKey	Lets you assign text to function keys.
IHelp	Gives you keyboard control over certain operations usually performed by the mouse, like enlarging or shrinking windows.
NoCapsLock	Temporarily disables the Caps Lock key.

Be sure to include the underscore after CX.

All of the Commodities programs share a common Tool Type, CX_PRIORITY = <n>, which assigns priorities to the Commodities Exchange programs. This priority is only relative to the other Commodities programs. All the programs are set to a default priority of 0. If you enter a Tool Type changing the priority to a higher value, that program will have priority over any other Commodities Exchange program.

For instance, IHelp and FKey both allow you to assign operations to function keys. If both programs have an operation assigned to F1, the program with the highest priority will intercept the key first, making it unavailable to any other Commodities programs.

There are two Tool Types that only apply to programs that open a window, such as Blanker and FKey. CX_POPUP = no prevents the program window from opening when the icon is opened. The program will be activated when you double-click on its icon, but its window will remain closed.

CX_POPKEY = <key> determines the **hot key** for the program. When the hot key (or key combination) is pressed, the program's window is automatically brought to the front of the screen. If the window is hidden, it will be opened. The hot key does not start a program.

Acceptable Key Combinations

When specifying key combinations for a Commodities Exchange program, you can use any of the function keys (F1 through F10) and any of the keys in the typewriter area of the keyboard (numbers, letters, symbols, etc.). However, keys from the typewriter area must be preceded by a qualifier. The allowable qualifiers are:

Qualifier	Key
Alt	either Alt key
RAlt	right Alt only
LAlt	left Alt only
Shift	either Shift key
RShift	right Shift only
LShift	left Shift only
LCommand	left Amiga
RCommand	right Amiga
Control	Ctrl
Numericpad	specifies a key on the numeric keypad
Rbutton	click the menu button
Leftbutton	click the selection button

Qualifiers can also be used before function keys, but it is not mandatory. You can use any combination of qualifiers, but it must be followed by a typewriter or function key. A qualifier is only recognized once in a combination, so a combination of:

LAlt RCommand LAlt F10

is the same as

LAlt RCommand F10

Some acceptable combinations are listed below:

Alt F6

LCommand 8

Control LShift Y

Leftbutton Control CapsLock =¹

Numericpad 8²

¹Click the selection button, then press Ctrl-Caps Lock =

²Press the 8 in the numeric keypad. The 8 in the typewriter area will not satisfy the combination.

When specifying key combinations, leave a space between the two keys. For instance:

CX_POPKEY = F9

CX_POPKEY = Shift F4

CX_POPKEY = LShift LAlt LCommand X

For a list of acceptable key combinations, see the chart on page 5-29.

AutoPoint

AutoPoint allows you to select windows without clicking the selection button. To start AutoPoint, double-click on its icon. AutoPoint does not open a window.

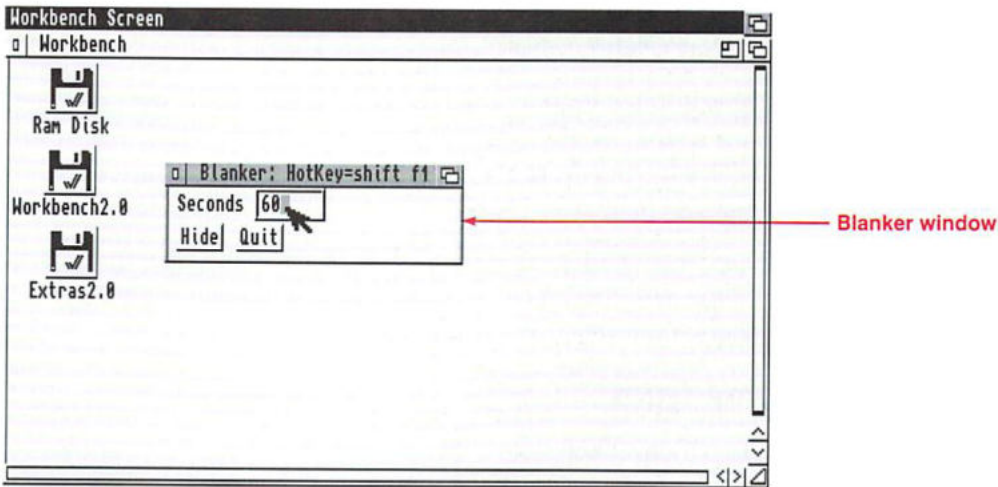
When AutoPoint is running, the system activates the window that is underneath the pointer. This eliminates the need to click the selection button.

The AutoPoint icon acts like a toggle switch. To exit AutoPoint, double-click on its icon again. You can also open the Exchange window, select AutoPoint from the scroll gadget, then select the Kill gadget.

Blanker

When the Blanker program is operating, the screen will automatically go blank if no input has been received during a specified period of time. This helps preserve your monitor. The default time is 60 seconds. If you do not press a key or click a mouse button during a 60 second period, the screen will go blank.

When you double-click on the Blanker icon, the following window appears:



To change the default time, select the Seconds text gadget, and enter the new value. To close the window, but not exit the program, select the Hide gadget. If you want to exit the program, select the Quit gadget. You can also choose the Hide and Quit menu items.

Tool Types

Blanker supports a SECONDS = <n> Tool Type that allows you to specify the number of seconds that will pass before the screen goes blank. For instance, to change the value to 30 seconds, enter SECONDS = 30.

The angle brackets indicate that information must be substituted. Do not type the brackets.

ClickToFront

ClickToFront allows you to bring a window to the front of the screen by double-clicking in it. You do not need to select the window's depth gadget.

To start ClickToFront, double-click on its icon. It does not open a window. (Remember, you can also put ClickToFront in the WBStartup drawer so that it is automatically started each time you boot.)

To exit ClickToFront, double-click on its icon again, or open the Exchange window, select ClickToFront from the scroll gadget, then select the Kill gadget.

Tool Types

ClickToFront supports a QUALIFIER Tool Type. This allows you to specify a qualifier key that must be pressed while you double-click in the window you want to bring to the front of the screen. There are four acceptable key arguments:

Lalt	Left Alt—Default
Ralt	Right Alt
Control	Ctrl
None	No key

For instance, if you have specified QUALIFIER = Lalt and ClickToFront is activated, you would hold down left Alt and double-click in the window you wanted to bring to the front of the screen. QUALIFIER = Lalt is the standard setting as supplied by Commodore-Amiga.

(This page was intentionally left blank.)

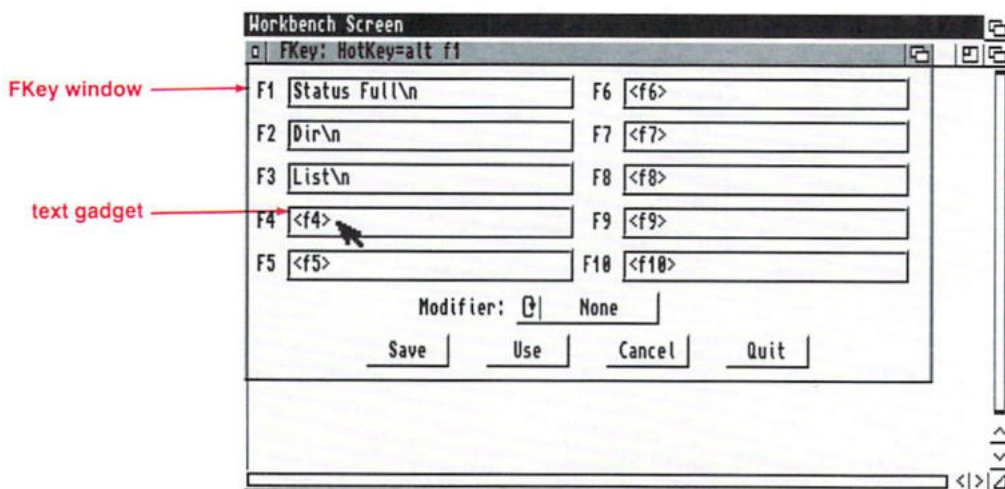
FKey

FKey allows you to assign text to a function key. This is of particular interest to frequent Shell users as it eliminates repetitive typing of AmigaDOS commands.



FKey assigns the function keys to the specified text before the system or any other application accesses the key. Therefore, if your application uses function keys, you should disable or quit FKey when using that software.

When you open the FKey icon, the following window appears:



You can assign text to every function key and Shifted function key, giving you a total of 20 assignments. Notice that the first three function keys are already assigned to AmigaDOS commands. These are Status Full, Dir, and List, respectively. (For an explanation of the commands see Chapter 8, "AmigaDOS Reference.") The output of these function keys is viewable through the Execute Command menu item or a Shell window. You can edit these text gadgets if you wish.

To enter text, select the appropriate text gadget in the FKey window, and type the text.

Some special characters supported by the text gadgets are:

- `\n` Adds a Return. If the function key is linked to an AmigaDOS command and the `\n` is omitted, the command will appear in the Shell window, but you will have to manually press Return.
- `\r` Adds a Return.
- `\t` Adds a Tab.
- `\0` Adds a zero.
- `<key>` You can use angle brackets to enter key combinations; the combination must be prefaced by a qualifier.

There are several gadgets at the bottom of the FKey window. The Modifier cycle gadget determines whether the text is attached to the function key alone or to the combination of Shift and the function key. When None is displayed, the text is assigned to the function key alone. When the gadget displays the word Shift, text is assigned to the Shifted function key.

The four action gadgets allow you to enable or quit FKey:

- | | |
|--------|---|
| Save | Enables FKey and permanently saves the text entered in the window. The text will be saved even after FKey is shut down. |
| Use | Enables FKey and temporarily uses the text in the window. When FKey is shut down, the current text will be lost, and FKey will revert to using any previously saved text. |
| Cancel | Enables FKey but ignores any recent changes made to the text. Only the last saved text is recognized. |

Quit	Shuts down the FKey window disabling all function key assignments until the program is run again. This is the same as choosing the Quit menu item.
------	--

The next time you start FKey, only the saved text will be present in the window.

Tool Types

The angle brackets indicate that information must be substituted. Do not type the brackets.

FKey supports a <Function Key> = <text> Tool Type which allows you to assign text to a function key. The acceptable values for <Function Key> are F1 through F10 and SF1 through SF10 for Shifted function keys.

The function key assignments mirror those made in the FKey window. If you enter text in the window, it will appear in the Tool Types gadget, and vice versa.

IHelp

IHelp allows you to use the keyboard, instead of the mouse, to perform certain operations usually performed by the window gadgets. To start IHelp, double-click on its icon. IHelp does not open a window.

The operations performed by IHelp, and the default function keys assigned to those operations, are listed below:

Cycle windows (Default—F1)	Brings the rearmost application window on the Workbench screen to the front of the screen and activates it. This only affects application windows opened by tools or projects, such as the Clock. Disk and drawer windows are not affected.
-------------------------------	---

Enlarge window (Default—F2)	Enlarges the active window to its maximum size, taking into account the edges of the screen.
Shrink window (Default—F3)	Shrinks the active window to its minimum size.
Cycle screens (Default—F4)	Brings the rearmost screen to the front of the display.
Zoom window (Default—F5)	Zooms the active window just as if you had selected the window's zoom gadget.

To exit IHelp, double-click on its icon again, or open the Exchange window, select IHelp from the scroll gadget, and select the Kill gadget.

Tool Types

You can change the default keys assigned to the IHelp operations by entering Tool Types in the IHelp icon's Information window. Each Tool Type takes a key argument. This argument is the key or key combination that you want to press to invoke the operation.

The acceptable Tool Types and the operations they correspond to are listed below:

CYCLE =	Cycle windows. For example, CYCLE = Shift C
MAKEBIG =	Enlarge active window. For example, MAKEBIG = Control Shift B
MAKESMALL =	Shrink active window. For example, MAKESMALL = RAlt S
CYCLEScreens =	Cycle screens. For example, CYCLEScreens = LShift Alt S
ZIPWINDOW =	Zoom active window. For example, ZIPWINDOW = RCommand Z

NoCapsLock

NoCapsLock disables the Caps Lock key. The Shift keys still function normally, but you don't have to worry about accidentally pressing Caps Lock while using the keyboard.

To start NoCapsLock, double-click on its icon. It does not open a window. To exit NoCapsLock, double-click on its icon again, or open the Exchange window, select NoCapsLock from the scroll gadget, and select the Kill gadget.

Chapter 6. Using a Hard Disk

This chapter provides details on using your hard disk. This includes information on HDBackup, a program used to back up and restore files on your hard disk, and HDToolbox, a utility for advanced users that allows control over hard disk operations.



Rather than using HDBackup to back up your hard disk, you may choose to use BRU, an advanced backup and restore utility accessed through the Shell. BRU is documented in Appendix C of this manual.

About Your Hard Disk

The following information is not essential for using a hard disk. You may want to read it to help understand more about what is going on inside your Amiga system.

Your hard disk allows you to store, use and retrieve large amounts of data quickly and conveniently. Information is stored on hard drive disks (or platters) located within the hard drive. Unlike floppy disks, these platters cannot be removed and are protected from the wear and tear of being handled.

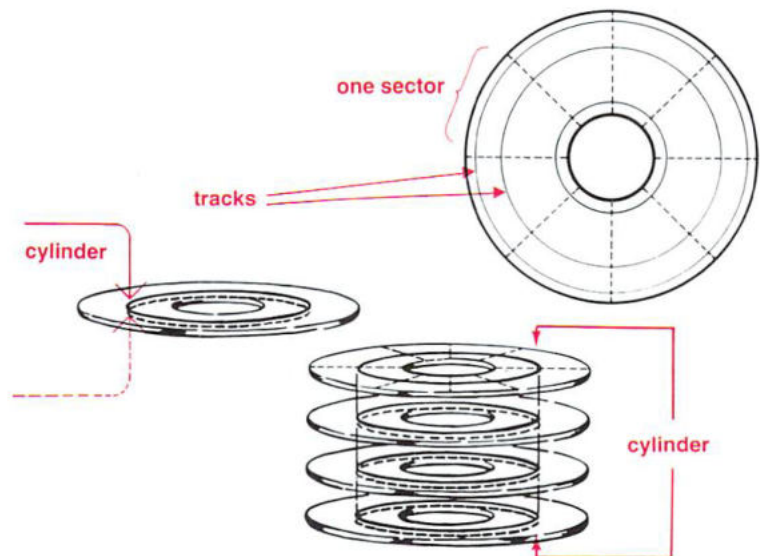
Hard disk platters look like small compact disks. They are coated with a magnetic recording surface that records information on both sides. A hard disk usually contains anywhere from one to eight platters.

In order for the computer to locate information quickly on the hard disk, the platters are divided into smaller sections. Each hard disk platter is organized into tracks, cylinders and sectors.

Tracks are similar to the grooves on a record album. They divide the hard disk platter into concentric circles.

A **cylinder** on a single hard disk platter is the recording space on both the top *and* bottom of *one* track. A cylinder on a multiple-platter hard disk is the recording space (top and bottom) on *all* of the platters of the tracks with the same track number.

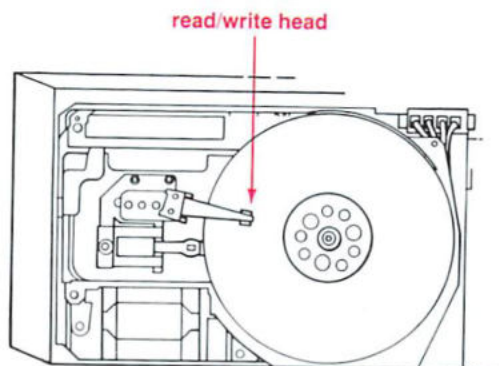
Sectors are like pie slices that divide the tracks. They are the smallest unit of storage on the hard disk platter, usually 512 bytes.



The hard disk finds information by using the combination of three locations—cylinders, tracks and sectors—as an “address”. It is similar to the box number, street and city on a mailing address.

Information stored on the hard disk platter is read and written by **read/write heads**. A read/write head is similar to the head of a cassette tape deck, but moves across the surface on an arm similar to the tone arm of a record player. The head discharges magnetic impulses at the right places to record data. To read data, the head(s) move to the appropriate cylinder and sense the

magnetic impulses from the sectors as the rotating platter passes beneath them. There is a separate head for each surface (top and bottom) of each hard disk platter.



There are many types of hard drives, with varying storage sizes from 20MB (over 20 million characters) to 500MB of information. A 50 MB hard drive has about as much storage space as 59 of the standard Amiga 3.5-inch floppy disks. Assuming an average page holds about 3500 characters, this would be the equivalent of nearly 14,400 typed pages.

Hard Disk Partitions

Because the storage capacity of hard disks is large, your hard disk can be divided into **partitions** which are simply subdivisions of the hard disk's storage space. Partitions can be used to better organize your work. For instance, you may want to separate the system software and other files your computer uses from your own application programs and files. If you add an additional operating system to your Amiga (such as the UNIX® operating system), you may decide to provide it with its own partition to keep it separate from AmigaDOS.

Commodore-supplied hard disks come with at least one partition—the Workbench (or System2.0). Depending on which computer you own, you may have received up to two additional partitions including the Work partition which is provided as your storage space.

With the HDToolbox program, you can partition your hard disk in any way you'd like, giving each partition any name you'd like (other than devices that already exist). This feature is described in detail in the HDToolbox section beginning on page 6-49.

The best time to partition a hard disk is before you begin using it. All information on affected partitions (for instance, a large partition you may have separated into two smaller ones) is erased in the process. If you already have information stored on affected partitions, you should back up the information and restore it after you partition. For more information see "Backing Up Your Hard Disk" on page 6-14.

Copying Software to Your Hard Disk

In addition to working with the utilities provided on Workbench, you will want to transfer applications from floppy disks to your hard disk. It is faster and more convenient than using floppy disks. Rather than inserting floppy disks each time you want to use a program, you can call up programs from the hard disk.

Most software can be installed on your hard disk. Many applications provide an easy-to-use installation process. Read the manual that is supplied with a program carefully, as well as any Read Me files on the disk. *If a program includes installation directions, you should always follow them precisely.*

In case you have a program without installation directions, this section demonstrates how to copy a generic software program to your hard drive. In this example, we assume you are installing a program to a partition on your hard disk called Work.

You should create a new drawer for each application you install and name it accordingly:

1. *Open the Work partition window by double-clicking on its icon.*
2. *Choose New Drawer from the Window menu.*

A new drawer icon called "Unnamed1" will appear in the window.

A requester will appear telling you to enter a new name for "Unnamed1" in a text gadget.

You can give a drawer any name you'd like, but you should use a name that reminds you of its contents, such as the name of the program. (For more information on renaming icons, see "The Icons Menu" section on page 2-67.)

3. *Using the Backspace key, delete the contents of the text gadget (or use right Amiga-X) and type in the correct name.*

Be sure to delete any spaces before or after the new name.

4. *Press Return or select the OK gadget.*

The requester will go away, and the new name will appear under the icon.

You can now copy the software into the new drawer. This can be done by simply pointing and dragging.

5. *Put the software disk into the disk drive.*

Make sure the software disk is write-protected. This is a precautionary measure to ensure that you don't alter your original software disk.

6. *Double-click on the disk's icon.*

Look at the contents of the disk on your screen. Often software disks contain duplicate files that are already on your system. You next need to copy any files *that are not already on your system* to the new drawer you created.

7. *Holding down the Shift key, point to each icon on the software disk that is not already on your system. Keeping the Shift key held down, drag the icons over the new drawer and release the selection button.*

You will notice that both your hard disk light and your disk drive light will begin flashing. This shows that Workbench is reading from your program disk and writing to your hard disk.

When the lights have stopped flashing, the information transfer is complete. Your software should now be in the drawer you created.

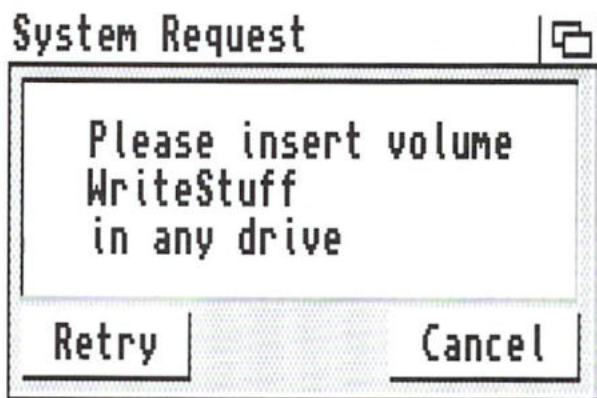
8. *Remove the original software disk from the floppy disk drive, and try to run the program from your hard disk.*

If the program runs properly from the hard disk, you have installed it correctly. If you have any problems running the software or if requesters appear asking for the original software disk, read the following section.

Troubleshooting

If you are having trouble running a program from the hard disk, you may have to set up an **ASSIGN statement** for the software to run properly.

Often, the indication that you need an ASSIGN statement comes in the form of a requester. For example, suppose you copied a program called WriteStuff into a drawer called WS in the Work: partition. When you try to run the software, a requester appears saying:



This means that although you have already installed the program on your hard disk, the program itself doesn't know where to look for the files it needs. Instead, it looks for them on the *original disk* (volume) on which it was distributed—in this example, the disk called WriteStuff. By using the ASSIGN statement, you will tell the system *where* to find the required files.

ASSIGN statements are AmigaDOS commands that are entered through the Shell. Chapter 7, "Using AmigaDOS," and Chapter 8, "AmigaDOS Reference," of this manual describe how to use AmigaDOS commands. If you are not at all familiar with using the Shell, you may wish to read Chapters 7 and 8 before attempting to use ASSIGN statements. The following instructions will, however, take you through the process of using ASSIGN statements.

Adding an ASSIGN Statement to your User-Startup File

The Shell is a window that you use to communicate with the Amiga via typewritten commands. In this case, you will use the Shell to enter the ASSIGN command in the proper place—the **User-Startup** file.

The User-Startup file is a list of commands that tell the computer what to do when it boots up. You will need to put the ASSIGN statement into the User-Startup file so that as soon as you boot your computer, your installed program will work.

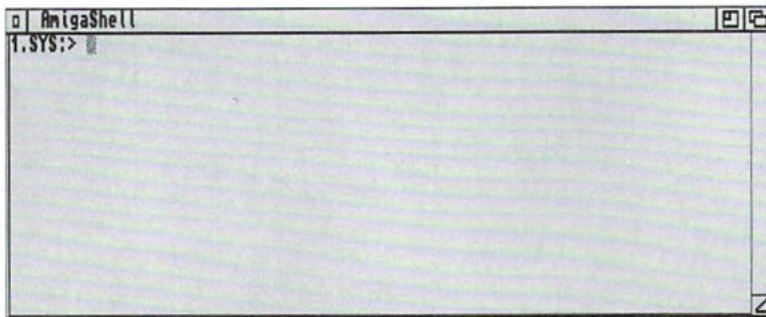
The User-Startup file is not on your disk as supplied by Commodore; however, you will create the file the first time you follow this procedure.

If a requester similar to the previous example has appeared, select the Cancel gadget on the requester. The requester may appear again. Select Cancel each time it does.

To add an ASSIGN statement to your User-Startup:

1. **Double-click on the Shell icon, which is normally located in the System2.0 window.**

The Shell window will appear.



The Shell gives you a text "prompt" which ends in a ">". This is where you will type in an AmigaDOS command followed by a Return. Please see the section "The Shell" on page 7-13 of this manual to familiarize yourself with the Shell window.

2. **Type the following words into the Shell after the prompt:**

ED S/User-Startup

then press Return.

Make sure you type in the command exactly as it is above.

This command tells the computer you will edit (create or make changes to) the User-Startup file which is located in the directory called S. ED tells the computer you will be using the text editor that is called ED. A text editor is a program that makes it possible to create and make changes in a text file.

You will notice that a new window will open. You are now using ED, and on the screen is the User-Startup file. The first time you use it, it will be empty. This is where you will always enter your ASSIGN statements. Move the cursor so it is at the beginning of a blank line.

ASSIGN statements are set up as follows:

```
ASSIGN diskname: partition:drawer (Return)
```

where:

ASSIGN	Is the first word you type in to tell the computer you are entering an ASSIGN statement.
diskname	Is the name of the disk which contained the software program. This must be entered in the exact form that was stated in the requester.
partition	Is the name of the hard disk partition which contains the drawer where you will keep your program.
drawer	Is the name of the drawer you made for the software.

Example ASSIGN statement

In the previous example, you copied a program called WriteStuff to a drawer called WS in your Work partition. You received a requester which said "Please insert volume WriteStuff into any drive". It is looking for the WriteStuff disk. In this case, your ASSIGN statement would read as follows:

```
ASSIGN WriteStuff: Work:WS
```

Now that you have seen *how* to form an ASSIGN statement, you should have a better idea of what it does. You are "assigning" the name of the original software disk to the name of the drawer that *now* contains the program. Using the previous example, when you double-click on the program's icon, instead of the program looking for the *volume* WriteStuff (which is the original software disk), it will look in Work:WS (where Work: is the partition and WS is the name of the drawer).

The following example illustrates an ASSIGN statement if the software drawer is within *another* drawer. If you copied the same program into the WS drawer which is *within* a drawer called Projects, your ASSIGN statement would read:

```
ASSIGN WriteStuff: Work:Projects/WS
```

NOTE: In order for the ASSIGN statement to work properly, you must make sure it is set up properly.

- Make sure that the disk name in the ASSIGN statement is the *exact* name of the software disk. This is usually the same as the name of the program, but not always. The best way to check is to insert the software disk and look for the disk's name under its icon.
- Make sure that the disk's name is followed by a colon.

- If the original disk name contains spaces, then that name (including the colon) must be enclosed in double quote marks.
 - Put one space after ASSIGN and one space after the disk name.
 - Use a colon between the partition name and the drawer name (and slashes between any drawers after that).
3. *Following the directions above, enter your ASSIGN statement in the blank line. Press Return.*
 4. *Press Esc (in the upper left corner of your keyboard).*

Your cursor is taken to an asterisk at the bottom of the screen.

The following step tells you how to *save* the changes you've made to the User-Startup file. (If you have made a mistake and want to exit the User-Startup file *without* saving your changes: Type a Q and press Return. If a requester appears saying edits will be lost, type a Y. Proceed to Step 6.)

5. *To save your changes in the Startup-sequence file, type an X and press Return.*

You are returned to the Shell window.

6. *In the Shell window, type:*

ENDSHELL

then press Return.

This command closes the Shell window.

To see if your ASSIGN statement worked, reboot your system (by pressing the Ctrl key simultaneously with both Amiga keys). Try opening the program from its icon on your hard disk. If you can use your software, you have used ASSIGN properly. If you used ASSIGN improperly (such as typing in the wrong

name of the software, partition or drawer) one or more of the following may happen:

- You receive a software failure message when your computer boots up. Press the left mouse button to reboot.
- Your Workbench icons do not appear, and a message is presented saying that it cannot find the software and the "assign failed". Use Ctrl-C to get to the Workbench screen. Open a Shell, and repeat the steps above to edit User-Startup. Look for what you may have entered improperly, correct it, and reboot.
- When you try to call up the program, you receive another requester telling you to "Please insert volume (*program name*) into any drive." Open a Shell, and repeat the steps above to edit User-Startup. Look for what you may have entered improperly, correct it, and reboot.
- If the program is not functioning properly, examine the new drawer (open the drawer and choose the Show All Files menu item) to see if it includes any of the following drawers:

Fonts	Devs/Printers
L	C
Libs	System
S	Utilities
Devs	Expansion
Devs/Keymaps	

If one (or more) of the drawers in the list exists in the new drawer, you must assign each to the corresponding device on your hard drive. The drawer most likely to be found in your new drawer is Fonts. Using the previous example, the following demonstrates ASSIGNing Fonts using the extended assign feature:

ASSIGN FONTS: WORK:WS/FONTS ADD

Backing Up Your Hard Disk

Although your hard disk is a good place to store information, system and hard drive failures do occur. However infrequent, these failures can destroy all of the information on your hard disk.

File loss is not always the computer's fault. Another cause of data loss on hard drives is human error, rather than hardware failure. Common errors include accidentally formatting a partition when you meant to format a different one, accidentally deleting files by mistyping the file names or selecting the wrong gadget, or simply deleting files that you thought you no longer needed and then discovering that you really did need them.

Making regular backups of your hard disk is a necessary precaution against these errors. A **backup** (or **archive**) is simply a copy of the information in your hard disk, usually stored on either floppy disks or magnetic tape (if you have a magnetic tape drive). If you ever lose information, you can easily restore it to your hard disk from your backup.

HDBackup

HDBackup is a backup and restore utility included on the Extras disk. On hard disk systems, you will find its icon in the Tools drawer.

HDBackup provides an easy way to back up your hard disk to floppy disks or tape, and then, if necessary, restore them to your hard disk.

You have many options with HDBackup. You can back up every file on your system or you can limit backups to a given directory. You can set HDBackup to back up only those files which have changed since the last time you've archived. Also, HDBackup can compress files to a smaller size so you can save space on your backup disks.



Using HDBackup for the First Time

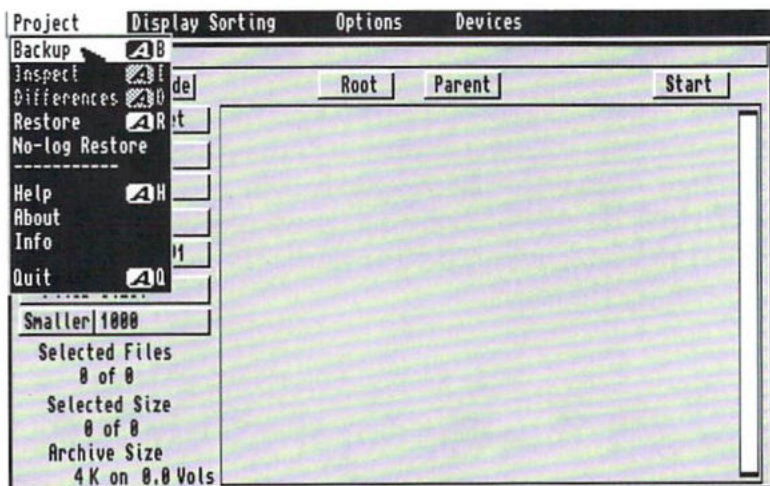
When you create a backup, you must first decide whether you want to back up an entire volume (a full backup) or just a portion of a volume (an incremental backup).

The following section will take you through the step-by-step procedure of creating a backup of an entire volume.

NOTE: Even if you plan to create incremental backups, use the following section, "Creating a Full Backup" as a tutorial the first time you use HDBackup.

Creating a Full Backup

1. Open the Tools drawer.
2. Double-click on the HDBackup icon.
The HDBackup screen will appear.
3. Choose Backup from the Project menu.



The Volume requester will appear with gadgets listing each of your devices and partitions, such as RAM:, WB_2.x:, DF0: and Work:. You now decide which of these volumes you would like to backup.

Directories and empty directories are indicated as such to the right of their names. Double-click on a directory to see what subdirectories and files a directory contains. To return to the previous directory, select the Parent gadget. If you have descended into a subdirectory and want to return to the original list of files, select the Root gadget.

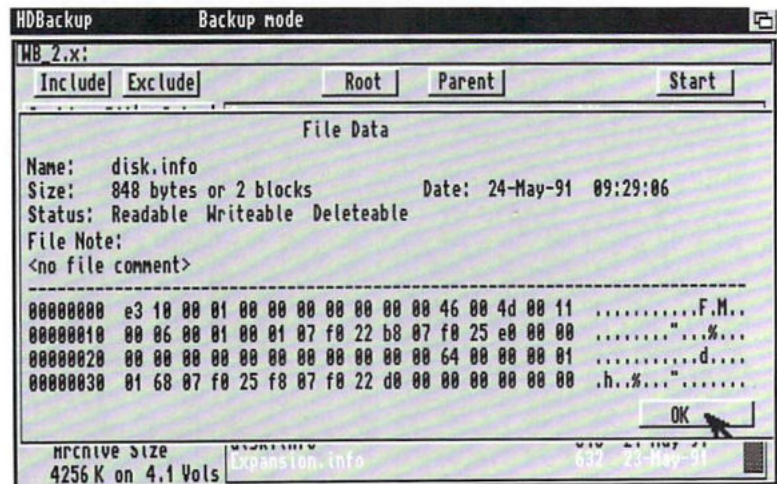
Files in the File Selection list include information on their size (in bytes), their last modification date (the last date you changed the file), and their archive bit status.

An archive bit tells you whether the file has been archived already. If a file list item has an "A" at the end, it means the archive bit is set. If there is no "A", the file has never been archived or has been changed since its last archive.

HDBackup has a File Data requester to provide you with even more information about a file in the File Selection list. To see the File Data requester:

5. Double-click on any file (not a directory) in the File Selection list.

A typical File Data requester looks like this:



This information tells you:

Name	The name of the file.
Size	The size of the file in bytes, and the number of archive blocks. A block is equivalent to a sector on the hard disk.
Date	The last modification date and time.
Status	A list of current attributes that have been set for the file. (For more information on attributes, see the "Information" section on page 2-78.)
File Note	Displays any comments attached to the file. (For more information on comments, see page 2-80.)

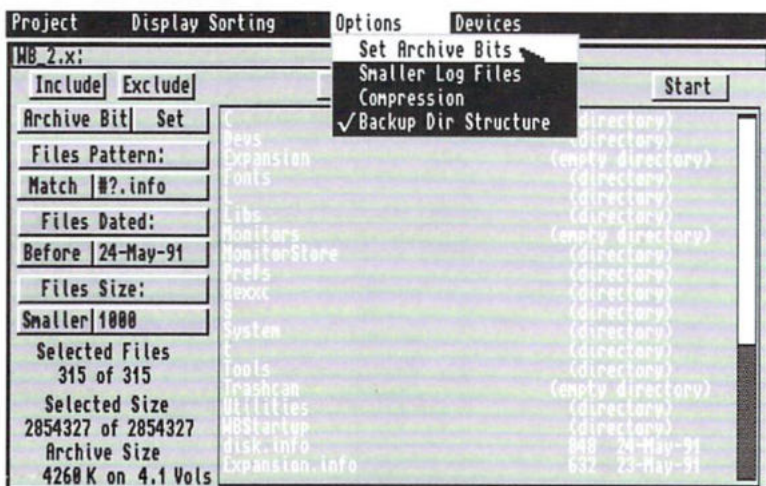
You will also notice four lines of letters and numbers on the bottom of the File Data requester. These are the first 64 characters of the file in **binary form** displayed in hexadecimal format (abbreviated "hex") which is how your computer reads the information. To the right of these lines is a column containing the first 64 characters of the file in text form. This can be used to take a quick look at the file's contents.

6. To exit the File Data requester, choose the OK gadget or type any letter.

Now you must decide if you want to set archive bits on the volume. If you wish to use archive bits to mark files as archived, you must choose a menu item. It is a good idea to set archive bits; in the future, you can create backups of only those files which have changed by using their archive bit status.

If you wish to set archive bits:

7. Choose Set Archive Bits from the Options menu.



A check mark next to the Set Archive Bits item indicates it is selected.

The File Selection list can be sorted in a number of ways. Your choices are shown in the Display Sorting menu:



A check mark indicates which method is selected.

With these features you can:

List Directories First Lists all directories before files. If this is not chosen, directories are listed with files according to the sorting method you selected.

You have the choice of one of the following sorting methods:

Sort by Name Lists files in alphabetical order. Files with numerical names are listed first.

Sort by Date Lists files chronologically according to the last modification date. In other words, the files that you've changed (or created) most recently will be at the bottom of the list.

Sort by Size Lists files in order of number of bytes, from smallest to largest.

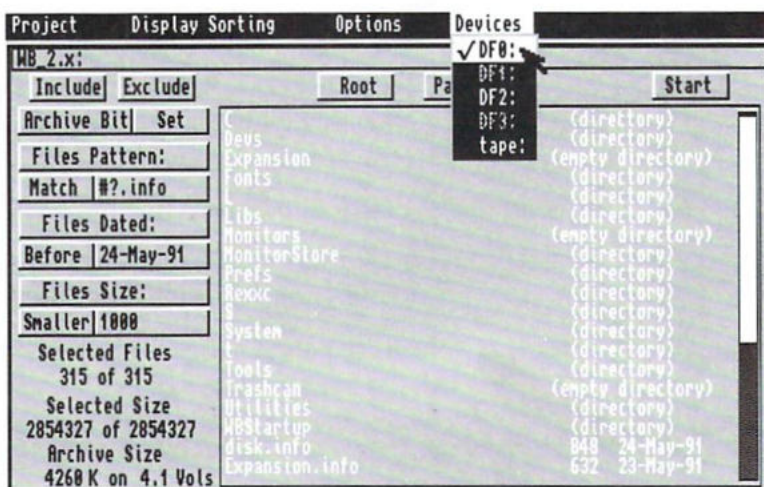
Sort by Archive Bit Lists files with a clear archive bit first.

If you wish to change the sorting method:

8. Choose a sorting method for the File Selection list from the Display Sorting menu.

Before you begin backing up the volume, you must tell HDBackup which device(s) will hold your backup. For example, drive DF0: is your archive device if you plan to archive with floppy disks to drive DF0:.

9. Choose the device you will use from the Devices menu.



To add a device (such as a tape drive) to the menu, see "Tool Types" on page 6-45.

To see how many disks the backup will require, you can refer to the Archive Size display in the lower left corner of the screen. The Archive Size display shows the total archive size in kilobytes. This is the size the backup will take up on disks or tapes and includes extra space for archive overhead (all the information the backup itself needs).

The Archive Size display also estimates how many disks you will need to archive the files you've selected. (It refers to disks as volumes.) This way you can have the proper amount of disks on hand. *Disks do not need to be formatted before using them with HDBackup.*

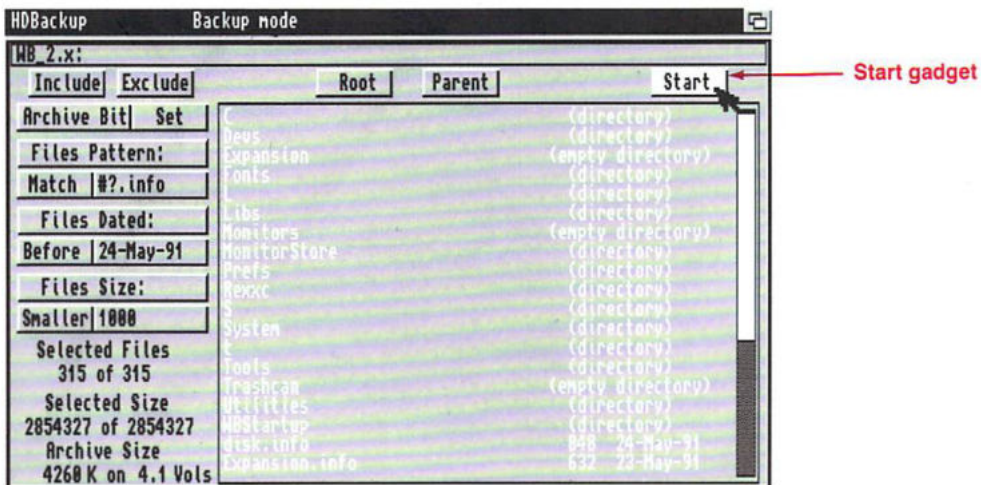
NOTE: When the device is set to tape, HDBackup does not know the size of tape you are using. The Archive Size display will still show the number of disks needed for the backup. Advanced users who want to use this estimate mode with tape backups must enter the tape size in Brutab. For information on customizing Brutab, refer to Appendix C, "Backing Up Your Hard Disk with BRU."

By default, HDBackup backs up the directory structure for each file. This means HDBackup will restore files to the same directory they were in when you created the backup. The check mark next to the Backup Dir Structure item in the Options menu shows this is set. Although you will normally leave it on, you can simply choose the item again to turn the option off.

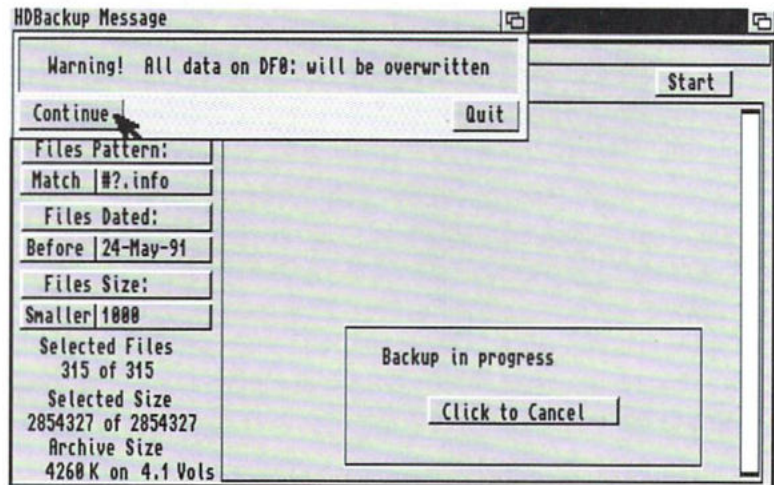
Once all the options have been selected, you must tell HDBackup to begin creating the backup:

10. Load the first volume (disk or tape).

11. Select the Start gadget.



HDBackup displays a requester warning you that all information on the specified drive will be erased.



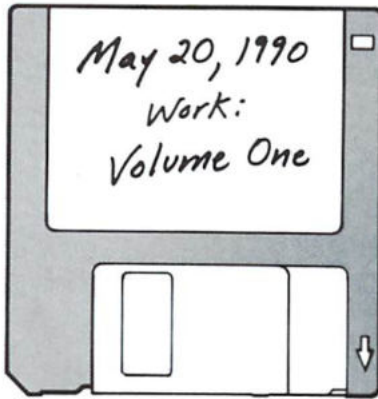
At this point, HDBackup waits for confirmation to proceed. Select the Continue gadget to proceed. To abort the Backup, select the Quit gadget.

12. To start backing up, select the Continue gadget.

HDBackup will prompt you to load disks to continue the backup.

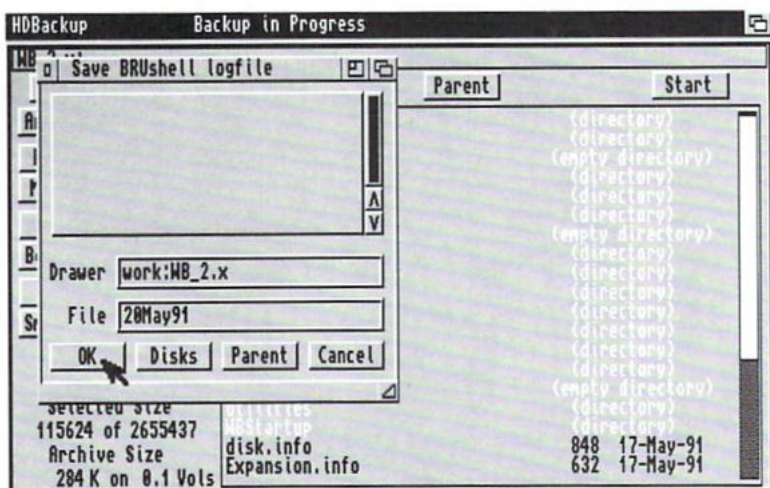
13. As you fill up each volume, label each disk with three pieces of information:
- The date of the backup.
 - The volume you backed up.
 - The backup disk's volume number.

For example:



It is essential to have an accurately labeled disk. When you restore files, you will need to know which disks to use, in their proper sequence.

When the backup is complete, the Log File requester will appear.



A log file is a report—or log—of what was included in your backup. While the actual backup is located on your backup disks or tapes, the table of contents of each backup is found on its log file. When you restore files, HDBackup uses the log file to provide you with the File Selection list for that particular backup. This allows you to exclude certain files from the restore process if you wish.

By default, log files are stored on the Work partition in a directory with the same name as the volume you are backing up. For instance, if you are backing up the Work partition, the file requester will show Work:Work as the default directory for the save. You need to create the Work:Work directory before trying to save the log file. If there is no Work:Work directory, the file requester will display Directory Error in the title bar. To create the directory, return to the Workbench screen, select the Work window, then choose New Drawer to create the directory. Return to the HDBackup screen, and select OK in the requester.

If you want to save the log file to a different location, simply type the correct volume name and any directory names in the text gadget.

Log files are automatically named according to the date that you've done your backup. For instance, a backup created on May 20, 1990, would have a log file called 20May90. (If you create a second backup of the same volume on May 20, 1990, it will be called 20May90.1.)

14. Select the OK gadget to store the log file in Work:WB_2.X.

If the Work:WB_2.X directory does not exist, you will have to create it.

If you do not want to save a log file with your backup, select the Cancel gadget to exit the requester. It is possible to perform a complete restore without a log file. However, you cannot exclude any files from the restore if you have not saved a log file.

Make sure to store your backup disks or tapes in a safe place.

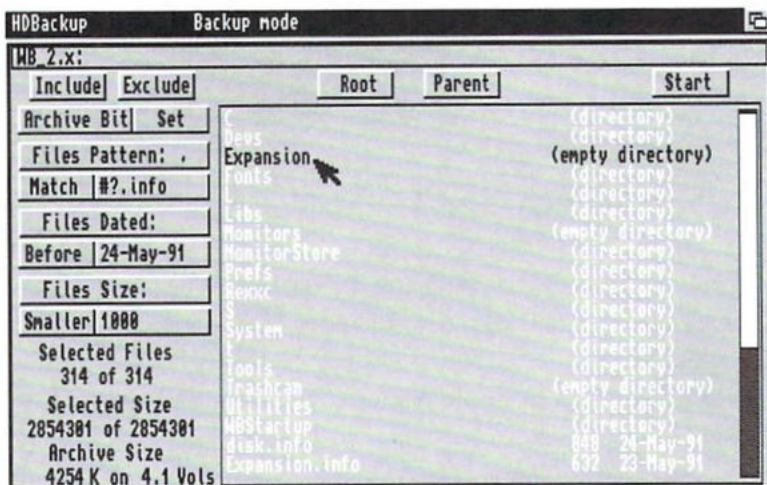


Creating an Incremental Backup

You may decide to exclude certain files from your backups because, for example, you have them on another disk, they might take up too much room on your disks, and so on. Now that you have seen how to back up a full volume, the following sections will explain the various ways to create incremental backups.

HDBackup is designed to make incremental backups very easy. Many of the gadgets you see in the HDBackup screen are used to select and exclude files from the backup, quickly and conveniently. The most basic way to exclude files is to point and click on the items that have been read into the File Selection list.

When you click on a file or directory in the list, it becomes shadowed.



This indicates that file has been excluded from the list and will not be included in your backup.

If you shadow a directory, the entire contents of that directory (including all of its subdirectories) will be excluded from the archive.

To exclude only certain files within a directory, you must descend into a directory (by double-clicking on it) and shadow files within it that you wish to exclude. (To get back to the previous list, select the Parent gadget.) The directory will no longer be shadowed, but all shadowed files within the directory will remain shadowed and excluded from the backup.

To deselect a file or directory which has been shadowed (and thus have it included on the backup), click on it again.

Selected Files and Selected Size Display

There is a running tally of selected files and selected size on the bottom left corner of your screen.

This lists the number of files you have selected from the list and the total size of the all these files in bytes. The number of selected files and the selected size decreases as you shadow files in the list.

Include and Exclude Gadgets

At times, you may want to back up only a few files of a very large volume. You could scroll through the list and shadow each file or directory to exclude, or you could use the Exclude gadget.

The Include and Exclude gadgets are at the top left corner of the screen.

The Include gadget tells HDBackup that *all* files in the File Selection list should be included in the backup. By default, HDBackup assumes that the Include gadget is selected. That is why all files which are read into the File Selection list are already *included* in the backup.

When you select the Exclude gadget, all files in the File Selection list are shadowed and *excluded*. All of the contents of directories are shadowed. You can then include some of the files by single clicking on them. To include files from a shadowed directory, you must descend into the directory and click on them. Try this and note the Selected Files display.

To return to the Include mode (and thus include all of the files again), select the Root gadget, then select the Include gadget.

File Selection Gadgets

You do not always need to point and click to exclude files. All of the gadgets on the left side of the HDBackup screen are used to include and exclude files from a backup (and later from a restore). These gadgets allow you to include or exclude files by their archive bit status, by a specified pattern that they may contain, by a specified modification date and/or by a specified size.

Here's how they work: File Selection gadgets are set *after* you have selected a volume and its contents are read into the File Selection list. You must activate the gadgets and provide them with specific information in their text gadgets. You must then select the Exclude gadget. All files that meet that criteria are shadowed and thus *excluded* from the backup.

All of the File Selection gadgets act in conjunction with one another. For instance, if you have activated both the Date Selection gadget and the Size Selection gadget, a file must meet the criteria of *both* to be included in (or excluded from) your backup.

The following four sections explain in detail how to use each of HDBackup's File Selection gadgets.

Selecting Files by Archive Bit Status

HDBackup can select files by the status of their archive bit. With this gadget, you can select a file list of only files with either a clear archive bit or a set archive bit, whichever you choose.

After a volume is selected and its contents have been read into the File Selection list:

1. Select the gadget marked Archive Bit.

This will highlight the gadget, indicating this feature has been selected.



2. Select either Set or Clear.

The Set gadget is a cycle gadget. With it, you can choose:

- | | |
|-------|--|
| Set | Select files with archive bits set. |
| Clear | Select files with a clear archive bit. |

3. Select the Exclude gadget

If you have chosen Set, all files with set archive bits will be shadowed (and thus excluded). If you have chosen Clear, all files with clear archive bits will be shadowed.

Selecting Files by Pattern

A **pattern** is simply a group of characters. HDBackup allows you to type in a pattern, and only files which contain that pattern (or only files which don't contain that pattern, whichever you choose) will be included in your File Selection list.

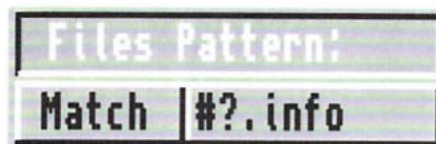
For example, you may want to archive every file in your DH0: partition, except for .info files. In this case, ".info" is the pattern you want HDBackup to exclude from your File Selection list.

The characters `#?` are used as a wildcard. The wildcard means "any text". For example, entering `#?.info` means any filename which ends in `".info"` will be included on the list. If you want to archive only files which begin with the letter "s", you would enter `s#?` in the text gadget. The wildcard must be used in the pattern unless you are selecting a pattern which would be matched exactly.

After a volume is selected and its contents have been read into the File Selection list:

1. Select the gadget marked Files Pattern:.

This will highlight the gadget, indicating it is activated.



Below this gadget is a cycle gadget that says Match. It has two functions:

- | | |
|-------|--|
| Match | Select files that contain that pattern. |
| ≠ | Select files that do not contain that pattern. |

2. Select either Match or ≠.

3. Click in text gadget, delete its contents and type in the pattern you wish to use.

4. Select the Exclude gadget.

Files which meet this criteria will be shadowed.

Selecting Files by Date

You can tell HDBackup to select files by date. It will then include on your backup only those files with modification dates before, after or exactly on the date you specify, whichever you choose.

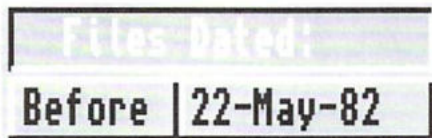
An example of this feature is performing a daily backup. As mentioned earlier, each file contains a modification date—this is the date and time a file was last used. In this case you want an archive of only those files which have changed today.

In this case you would tell HDBackup to exclude files dated *before* today.

After a volume is selected and its contents have been read into the File Selection list:

1. *Select the gadget marked Files Dated:.*

This will highlight the gadget, indicating this feature has been activated.



The gadget marked Before is a cycle gadget. With this gadget, you can select:

- | | |
|--------|---|
| Before | Select files with modification dates before the specified date. |
| After | Select files with modification dates after the specified date. |
| On | Select files with that exact modification date. |

2. *Use the cycle gadget to choose Before, After or On.*

Next you must set the date.

3. *Click in the text gadget, delete its contents and type in the date you choose.*

The date should be in this form:

DD-MMM-YY

where:

DD Is the day of the month.

MMM Is the name of the month,
abbreviated to its first three
characters (such as SEP for
September).

YY Is the last two digits of the year.

It is not necessary to use a leading 0 for single digits.

4. *Select the Exclude gadget.*

Files which meet the criteria you've set will be shadowed.

Selecting Files by Size

You may choose to back up files by a specific size. For example, you may want to back up only your very large files (such as data base files) that took you a long time to create.

After a volume is selected and its contents have been read into the File Selection list:

1. *Select the gadget marked Files Size:.*

This will highlight the gadget, indicating this feature has been selected.



Next you must specify a size.

2. *Click in the text gadget, delete its contents and enter the size.*

Size is specified in bytes.

The gadget that says Smaller is a cycle gadget. With this gadget, you can select:

Smaller	Select files smaller than the specified size.
Larger	Select files larger than the specified size.
Equal	Select files equal to the specified size.

3. *Using the cycle gadget, select either Smaller, Larger or Equal.*
4. *Select the Exclude gadget.*

Files which meet this criteria will be shadowed.

Smaller Log File Option

Normally the log file contains the complete File Selection list for that particular archive. This includes files which were included in the archive as well as files which were excluded by shadowing them. When you go to restore files, you will notice that Excluded files are represented in a different color.

By choosing Smaller Log files from the Options menu, HDBackup will eliminate the excluded files from your log file. This way, when you go to restore files, only files which were included in the archive will appear in the File Selection list. This option is useful for saving disk space if you have many log files saved on your HDBackupLogs disk.

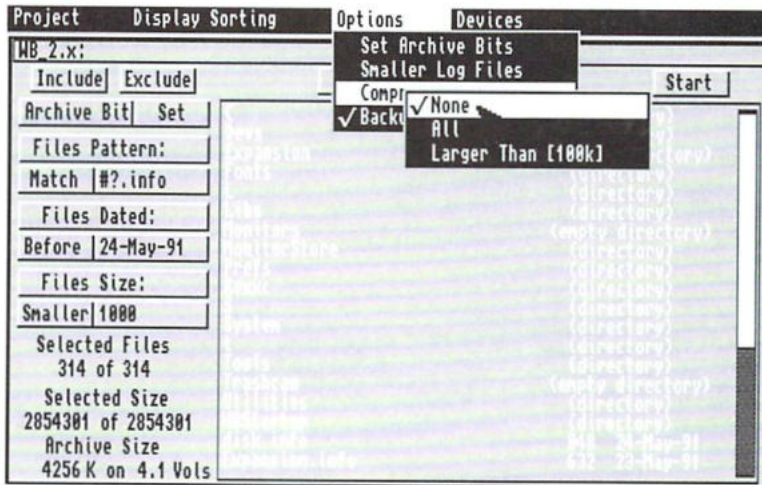
File Compression Option

The File Compression option was designed to save space on your backup disks. Files are **compressed**—made smaller—so more files will fit on each disk.

The smaller size of the archive is the advantage of compressing files. You should weigh this advantage with File Compression option's two disadvantages. The first is that it takes longer to perform the backup—sometimes two to three times longer—because for each file, HDBackup must first compress it and then save it to your backup disk. The second disadvantage is that HDBackup cannot provide you with an estimate in the Archive Size display area. Instead it reports “???” in that field. HDBackup would actually have to go through and compress each file just to give you an estimate.

To save backup disk space, you may be willing to overlook these disadvantages.

The Compression menu item gives you three choices.



These choices are:

- | | |
|----------------------------|---|
| Compress
None | By default, the File Compression option is set at None where no files will be compressed. |
| Compress
All | Every selected file on your File Selection list will be compressed. |
| Compress
Larger
Than | Only files larger than the specified size (in bytes) will be compressed. This option compresses files which are very space consuming, but cuts down the archive time by not compressing all of the files. |

When you choose the Compress Larger Than menu item, a requester will appear asking for the size (in bytes). A default size of 100 (K) will appear in the text gadget. If you wish to use this size, select the OK gadget. If you want to change the size, click in the text gadget and delete its contents. Enter the new size in the text gadget and select OK.

Files which have been compressed can be restored in the same way as regular files. HDBackup will automatically decompress files before it restores them.

File compression on a backup does not affect your original file in any way.

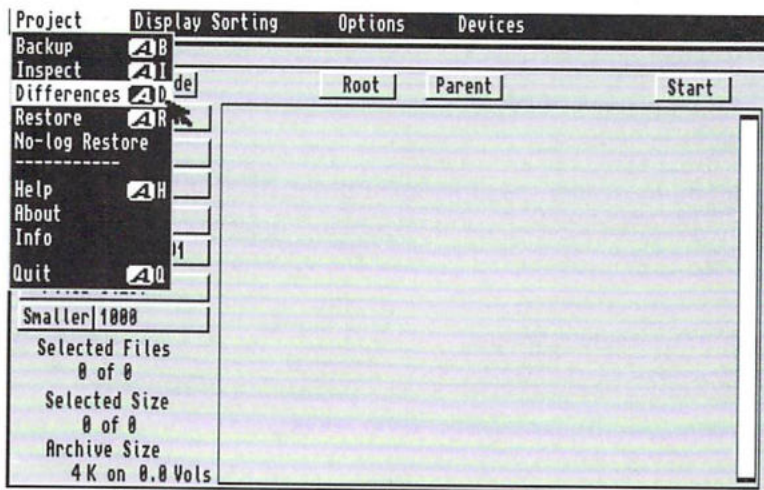
Checking Differences

After you've created a backup you may want some assurance that it is complete and error-free so it can be used in the future. The Differences mode checks a backup for errors and any differences between the files on your backup and the files with the same name on your hard disk.

The Differences mode can be used immediately after you've created a backup to make sure that the backup is complete—you should not see any differences between your backup and your hard disk files of the same name. You can also run Differences on a backup at a later time to see which files have changed since the archive was created.

To run the Differences mode:

1. *Choose Differences from the Project menu.*



The Log File requester will appear and show a list of volumes you have backed up.

2. *Click on the directory for the proper volume. From the log file list, click on the log file you will check for differences. Then select the OK gadget.*

The File Selection list on the HDBackup screen will display the list of files that you backed up.

3. *Check that the Devices menu is set properly to reflect which device(s) will contain your backup.*
4. *Insert the first backup disk.*
5. *Select the Start gadget.*

The volume requester will appear asking you to select the volume to which you'll be comparing the backup.

6. *Select the volume and then select the OK gadget.*

HDBackup will scan the list of files and report any errors and differences in files. These differences may include variations in file size, modification dates, contents, protection bit settings and file names.

If you have checked differences in the file immediately after creating it and HDBackup reports nothing, you know that there are no errors and none of the files have been changed since you archived. In this case you know your backup was complete and error-free.

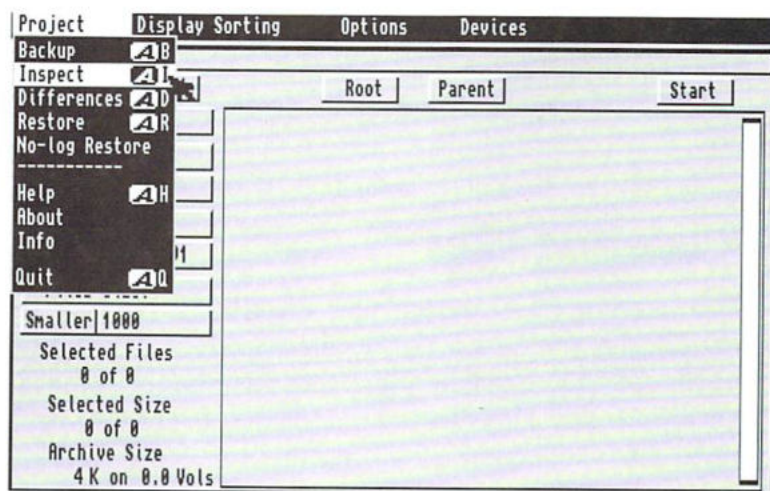
If files you backed up produce error warnings (such as a warning telling you a file may be corrupted), you must decide if the file is critical. If so, you may decide to investigate the problem and try to archive that file again on a separate disk.

Inspecting a Backup

In one sense, HDBackup's Inspect mode works similarly to the Differences mode—it reports errors. The Inspect mode does not compare a backup to files on your hard disk so it is useful if you want to check an older backup for errors.

To use the Inspect mode:

1. *Choose Inspect from Project menu.*



The Log File requester will appear and show a list of volumes you have backed up.

2. Click on the directory for the proper volume. From the log file list, click on the log file you will inspect. Then select the OK gadget.

The File Selection list on the HDBackup screen will display the list of files that you backed up.

3. Check that the Devices menu is set properly to reflect which device(s) will contain your backup disks.
4. Insert the first backup disk.
5. Select the Start gadget.

The volume requester will appear asking you to select the volume or directory to inspect.

6. Select the volume and then select the OK gadget.

HDBackup will start reading the backup and inspecting it. File names will appear in the screen's display area as it scans.

As in the Differences mode, HDBackup will display any errors it has found in the files.

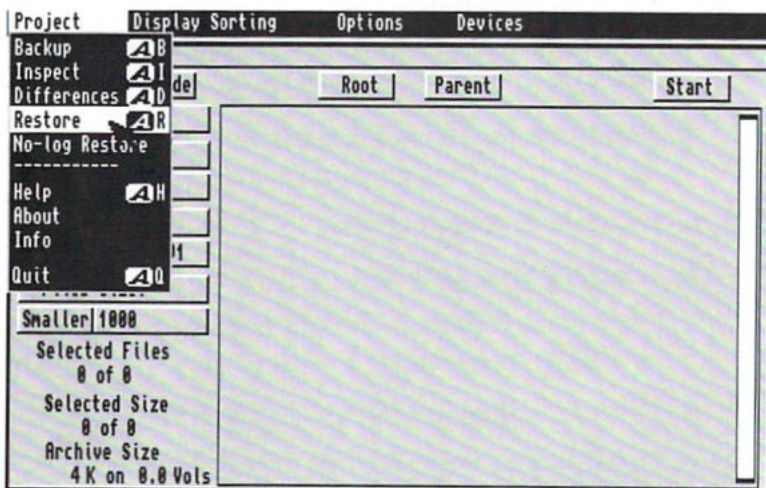
If no error messages appear, your backup is error-free. If error messages do appear, you may want to investigate the problem and decide if you want to try to back up the files again.

Restoring Files

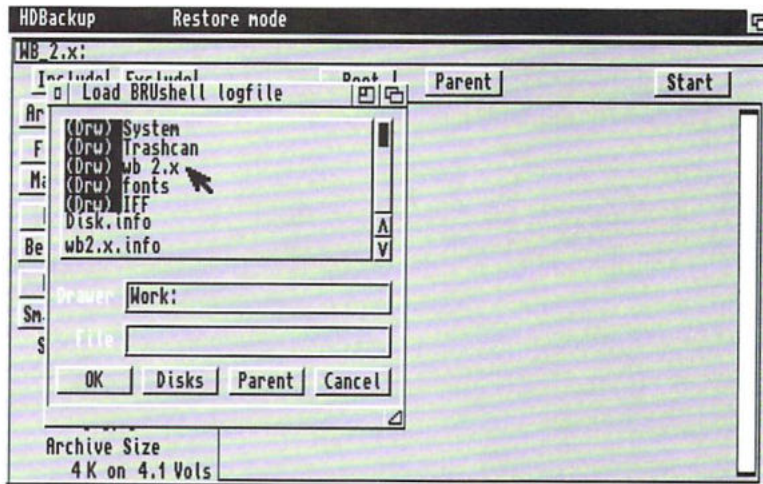
There are two ways to restore files. If you saved a log file with your backup, you can use the Restore menu item to look at the log file and choose exactly which files you want to restore. If you did not save a log file, you can use the No Log Restore menu item to perform the backup. However, in this case, you must restore all the files on the backup disks; you cannot exclude any files.

Restoring Files with a Log File

1. Choose the Restore item from the Project menu.



The Log File requester will appear and show a list of volumes you have backed up.



Double-click on the volume you want to restore to see its log files. If you made a backup of a directory within that volume, it will appear with all of the log files for that volume. You must double-click on that directory to see its log files.

- 2. Click on the volume you wish to restore. From the log file list, click on the log file which contains the file(s) to restore. Then select the OK gadget.**

The File Selection list on the HDBackup screen will display the list of files that you backed up. Remember, files in light blue were excluded from the backup when you created it. If the file(s) you wish to restore are not on the current log file, choose Restore from the Project menu. The Log File requester will appear, and you can search through another log file.

You may not wish to restore all of the directories and files that now appear in the File Selection list. You can now exclude files you don't want restored by shadowing them or by using the File Selection gadgets with the Include and Exclude gadgets as you did when creating a backup, explained earlier in this chapter.

3. *Exclude any files you do not wish to restore.*
4. *Check that the Devices menu is set properly to reflect which device(s) contains your backup files.*
5. *Insert the first backup disk (or tape).*
6. *Select the Start gadget.*

The volume requester will appear. This requester is similar to the requester which appears when you create a backup. You must tell it where you want the files restored.

Notice that the text gadget in the requester already contains a default—the volume that originally contained the files. If you want them restored to the same place you can simply select the OK gadget.

If you want files to be placed in another partition, click on the gadget which contains the name of that partition. If the files should be placed in a directory within a partition, you must type the partition name and the directory in the requester's text gadget. (For example, if you want to restore the file(s) to a directory named Junk in the Work: partition, you would type Work:Junk in the text gadget.)

7. *Select the volume you will restore to by clicking on a gadget or typing the name in the text gadget. Select the OK gadget.*

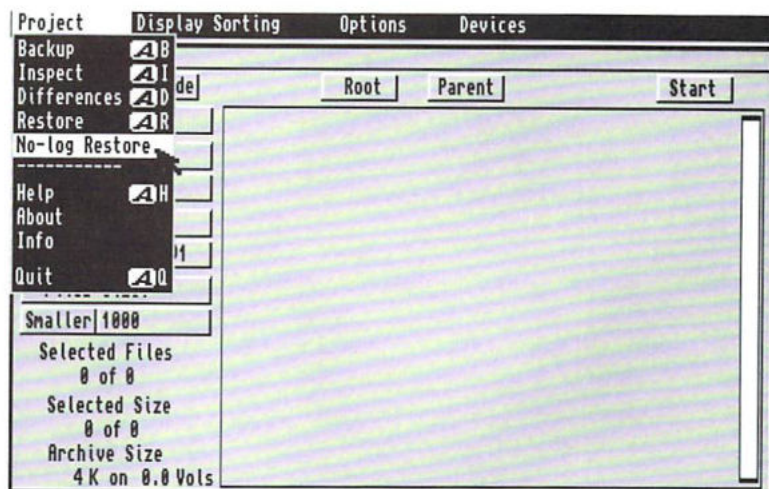
NOTE: Sometimes you may restore files from a backup which are already on your hard disk. HDBackup will not overwrite a newer hard disk file with an older version of the file from the backup.

You will see in the display area on the screen that HDBackup is scanning the backup and restoring the files. The flashing lights between your hard disk and your disk drive show that the information is being passed from one to the other. When the flashing lights have stopped, your restore is complete.

Restoring Files without a Log File

To restore files to your hard disk when you have not saved a log file:

1. Choose *No Log Restore* from the *Project* menu.



The Volume requester will appear so that you can choose the volume and directory where you want the files to be restored. The text gadget usually contains the name of the volume that originally contained the files. If you want them restored to the same place, you can simply select the OK gadget. Otherwise, select the gadget for the appropriate partition. You can also type the partition and directory name in the text gadget.

2. Once the correct partition and/or directory is displayed in the requester's text gadget, select the OK gadget.

If you have not already inserted the first backup disk or tape, a requester will ask you to do so. Select the Proceed gadget to continue. A requester will notify you when the restore is complete.

(This page was intentionally left blank.)

Tool Types

Besides the available options on the HDBackup screen, you can set options for HDBackup through Tool Types. Instructions for adding and changing a program's Tool Types are found in the "Tool Types" section on page 4-26.

As shipped, HDBackup is set to certain default Tool Types. If your choice is the default, you do not need to enter anything in the Tool Types window. If you do wish to change the defaults, the recognized KEYWORDS are:

- | | |
|---------|---|
| BRU | HDBackup uses the BRU program as its executable. (For more information on BRU, see Appendix C.) This option allows an alternate for the BRU executable to be given.

Default: BRU = bru |
| BRUPATH | Allows you to specify an explicit path to find BRU. <i>This path is limited to 64 characters and may not contain any spaces.</i> If this option is not used, HDBackup will search for BRU first in the directory containing HDBackup, and then in the C: directory.

Default: BRUPATH = C:
Example: BRUPATH = HDutils: |
| BRUARGS | Allows some user-specified arguments to be passed to BRU by HDBackup. (This is not very useful for the most part, except for debugging purposes.)

BRUARGS = <arguments> |

BRUSTACK	<p>Allows a specific stack size to be set for the BRU process by HDBackup. The default is to use the same stack size as HDBackup if BRUSTACK is not given.</p> <p>Default: BRUSTACK = 40000</p>
DEVS	<p>Allows you to add a device to the Devices menu (in addition to the pre-existing DF0: through DF3:). A total of four devices can be added to this menu. <i>This must come before a USE = option statement.</i></p> <p>Example: DEVS = tape: Example: DEVS = worm: tape:</p>
USE	<p>Specifies the names of the backup device(s) to use. These devices should already exist on the device menu (DF0: through DF3:), or should have been defined in the DEVS = option.</p> <p>USE = <name> [name] [name] Example: USE = df0: df1: df2:</p>
FONTNAME	<p>Sets the font that HDBackup will use.</p> <p>Default: FONT = topaz</p>
FONTSIZE	<p>Sets the size of the HDBackup font. The font can be any height from 6 to 15, <i>but the width must be 8</i>. This option precludes use of proportional fonts.</p>
SCREEN	<p>Sets the type of screen to use, either the Workbench screen or a custom screen.</p> <p>Default: SCREEN = custom SCREEN = workbench</p>

NUMCOLORS Allows you to choose the number of colors for a custom screen, either 4 or 8. (It has no effect if SCREEN = workbench.)

Default: SCREEN = 4

SCREEN = 8

LACE Controls whether or not the custom screen is (USA) 200 lines high or (USA) 400 lines high. This will also affect the colors used (see LACECOLORS).

Default: LACE = off

LACE = on

COLORS Sets the colors for a custom HDBackup screen. The number is of the format Red-Green-Blue, in hex. For example, black is 000 and white is FFF. *All colors must be specified if this parameter is used.*

Default: COLORS = 555,FFF,579,0FF,AAA,
0A8,B75,CCC

LACECOLORS Sets the colors for the custom screen that will be used if LACE = on is specified. See COLORS.

Example: LACECOLORS = 000,458,FFF,800,
7FF,AA3,8B2,111

BACKUP
RESTORE Causes HDBackup to automatically go into the desired mode, either Backup or Restore, and begin building the File Selection list.

BACKUP = <volume[path]>

RESTORE = <volume[path]>

START	<p>If set to on, the option set by specifying the BACKUP = or the RESTORE = option will begin automatically. Otherwise, the File Selection list is built, and the user must start the backup (or restore).</p> <p>Default: START = off START = on</p>
ICONS	<p>If set to on, log files will have icons created for them, as will any directories that need to be created in which to save the log file.</p> <p>Default: ICONS = on ICONS = off</p>
FILEICON	<p>Names the icon to use as a template for the log file icons. The .info extension will be added. If this is not specified, the internal default icon will be used. This has no effect if the ICONS = off option has been set.</p> <p>Example: FILEICON = work:my_icon</p>
DIRICON	<p>Names the icon to use as a template for the log file directory icons. The .info extension will be added. If this is not specified, the internal default icon will be used. This has no effect if the ICONS = off option has been set.</p> <p>Example: DIRICON = work:my_directory_icon</p>
LOGDIR	<p>Sets the directory in which to store the log files.</p> <p>Default: LOGDIR = sys:hdbbackuplogs</p>

HDToolbox

HDToolbox provides a variety of tools for controlling hard disk operations. HDToolbox is compatible with the A3000, the A2091, and other hard disks that follow the Amiga Rigid Disk Block standard.

To run HDToolbox from your hard disk:

1. *Open the System2.0 partition.*
2. *Open the Tools drawer.*
3. *Double-click on the HDToolbox icon.*

If your hard disk has not been installed:

1. *Boot your system with your 2.0Install disk.*
2. *Double-click on the 2.0Install icon, then open the Tools drawer.*
3. *Double-click on the HDToolbox icon.*

The first screen which appears is called the Hard Drive Preparation, Partitioning and Formatting screen. You will learn more about this screen in the next section.

You will use this opening screen to get to the functions of HDToolbox. The functions you can select with HDToolbox include:

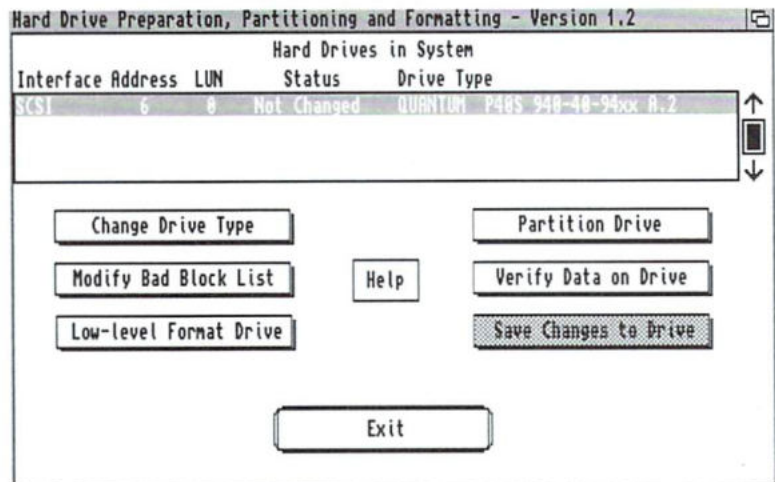
- Partitioning
- Preparing a New Hard Disk
- Configuring the Drive Type of the Hard Disk
- Low-Level Formatting a Hard Disk
- Locating Bad Blocks on a Hard Disk
- Defining File System Characteristics
- Adding, Deleting or Modifying File Systems

Detailed explanations and instructions for using each of these functions of HDToolbox are found in the remainder of this chapter.



The Hard Drive Preparation, Partitioning and Formatting Screen

The Hard Drive Preparation, Partitioning and Formatting screen provides a list of the hard drives you have connected to your system. This window can display up to four drives at a time. If there are more than four drives connected to a system, you can scroll the contents of the window by using the scroll bar or the scroll arrows at the right side of the window.



The information provided in the window is:

Interface

Displays the type of hard disk. The type of hard disk is almost always SCSI.

SCSI stands for Small Computer System Interface. A SCSI hard disk is part of a family of SCSI devices which connect to your computer. SCSI devices have a standard

connector, developed to be compatible with many different types of computers. SCSI devices are useful because they increase the speed and flexibility of your computer system.

The Interface will read XT if you have connected a A590 Hard Drive Plus.

Address

Displays a value, 0 through 6, that you have set for each SCSI device attached to your system. The address is used by the computer to find the information at this location. Each SCSI device on your system must have a different address.

The SCSI controller pre-installed in your computer is set as device 7. Most hard drives come pre-configured as device 0, so the first additional SCSI hard disk can be connected without change. If you add additional drives, you will need to reset them to different (unused) addresses. If two or more devices are jumpered to the same address, the system will not function properly. See the hard disk's documentation for more information, including jumper location.

LUN	<p>Shows the Logical Unit Number (LUN) of the drive, a value from 0 to 7. The LUN is a secondary address. It is used when a device controls multiple devices. For example, a controller card may be capable of controlling more than one hard disk. Just as each SCSI device attached must have a different address, each device attached to the controller card needs a different LUN.</p> <p>The LUN of a SCSI hard drive will usually be 0. See the controller card's documentation for more information on whether your controller supports multiple LUNS, and/or how to change them.</p>
Status	<p>Shows whether or not you have made any changes to a drive that have not been saved. To save changes after any HDToolbox operation, you must select the Save Changes to Drive gadget on the Hard Disk Preparation, Partitioning and Formatting screen.</p>
Drive Type	<p>Shows the drive's manufacturer, name, and revision.</p> <p><i>NOTE:</i> This information may not correspond exactly to the name and number listed in the drive's documentation. This was the name reported by the drive to HDToolbox.</p>

If the drive is listed in this window as Unknown, it is not partitioned and the drive type will have to be selected. (See "Changing the Drive Type" section on page 6-80.)

SCSI tape drives are always listed in Drive Type as Unknown.

The remaining gadgets are used to perform the functions of HDToolbox, described throughout the remainder of this chapter.

Because many functions of HDToolbox involve erasing all the information stored on your hard disk, the entire contents of your hard disk should be backed up before using the program. This may be done through the HDBackup program, described earlier in this chapter, or (for advanced users) through BRU, described in Appendix C.

You may abort any changes made to HDBackup screens or requesters by selecting the Cancel gadget. This gadget is present on each screen and will return you to the previous screen without saving any changes.

Partitioning

A partition is a subdivision of the space on your hard disk. Each partition on your system appears as an icon on the opening Workbench screen. Depending on which model computer you own, your hard disk may already be partitioned into sections called System2.0 and Work.

HDDToolbox provides a screen where you can easily partition your hard disk. You may have just installed a new hard disk and need to partition it. Or you may decide to repartition your existing hard drive.

Reasons for partitioning include:

- Organizing your work. You may want to keep certain files in their own partition so you can find them easier.
- Keeping AmigaDOS separate from a new operating system you've added (such as the UNIX operating system).
- Reducing fragmentation of partitions for a more efficient system.



The best time to partition the hard disk is before you begin using it. Whenever you partition your hard disk, all information stored on the altered partitions is erased. Before you begin, back up your entire hard disk with the HDBackup program (described earlier in this chapter) or with BRU (described in Appendix C). Your data can then be easily restored when you're finished.

Advanced users can use HDDToolbox's Advanced Options to modify the number of sector cache buffers, change file systems on the partitions and set boot priorities on bootable partitions. The Advanced Options are described later in this section.

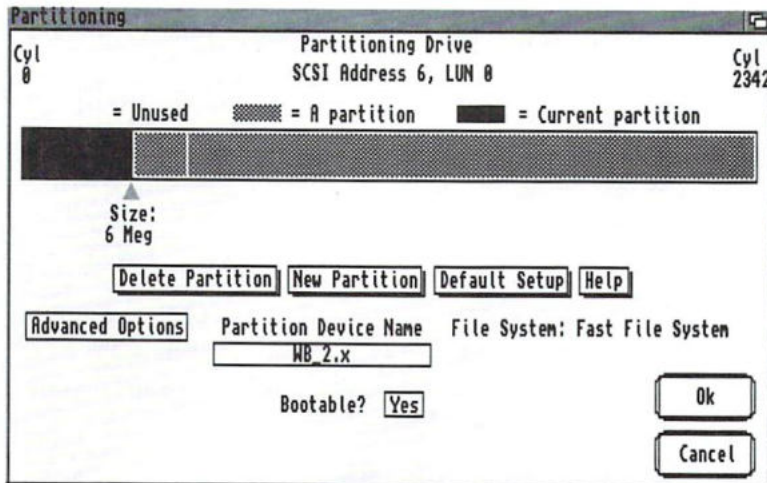
To partition, begin in the Hard Drive Preparation, Partitioning and Formatting screen:

- 1. Select the hard disk that you want to partition by clicking on its name in the list.***

The selected drive will be highlighted.

2. Select the Partition Drive gadget.

You will be taken to the Partitioning screen.



As you see, the space on your hard drive is displayed as a horizontal bar (the partitioning bar) with the number of the last cylinder shown in the upper right corner of the screen. With the default Workbench colors, the current (or selected) partition is displayed in black, with the size of the partition listed below it. Other partitions are shaded, with dark lines as divisions. Unused areas are displayed in solid gray. You will use the partitioning bar and your mouse to form your new partitions.

Before you begin to partition, note the following on the Partitioning screen:

Partition Device Name	Displays the name of the selected partition. To select a partition, click on it within the partitioning bar.
-----------------------	--

File System:	Displays the file system of the selected partition. (For more information on file systems, see "Modifying File Systems" on page 6-87).
Bootable?	Displays whether or not the selected partition can be used to boot the system. The Bootable? gadget defaults to Yes for the first partition and No for all other partitions.

Because partitioning is based on your personal choice of how you want your system, we cannot provide a step-by-step procedure. Instead, read the information below and form your partitions in any way you'd like.

Adjusting the Size of a Partition

1. *Click and hold the left mouse button on the blue triangle under the partition bar.*
2. *Slide the triangle to the new position and release the button.*

You cannot expand the size of a partition over a partition which already exists.

The new size of the current partition will be displayed under the triangle.

Sliding a Partition within the Partitioning Bar

You will need to slide partitions to make room for a new partition or to consolidate portions of unused space. Partitions will only slide on unused space in the partitioning bar. They will not slide over existing partitions.

1. *Click and hold the selection button on the partition you wish to move.*

2. *Still holding the button, slide the partition to where you want it and release the mouse button.*

If a partition is too small to conveniently click on, you can also move left and right through the partitions by using the left and right cursor keys, respectively.

Adding a New Partition

You cannot make a new partition over a partition which already exists. If all of the space on your hard disk is in use (i.e., no unused space in the partitioning bar), you will first need to make space for a new partition by making existing partitions smaller (see "Adjusting the Size of a Partition"). Unused space is shown as a gray block.

1. *Slide existing partitions so you have a solid block of unused space within the bar for your new partition.*
2. *Select the New Partition gadget.*

The New Partition gadget will be highlighted.

3. *Click on the unused portion of the partitioning bar where you want your partition.*

A new partition will fill the previously unused space. The Partition Device Name text gadget will display CHANGE_ME.

Renaming a Partition

1. *In the partitioning bar, click on the partition you wish to rename (making it the current partition).*
2. *Click in the Partition Device Name text gadget that displays the name of the current partition.*
3. *Delete the existing name, type in the new name and press Return.*

Deleting a Partition

1. In the partitioning bar, click on the partition you wish to delete (making it the current partition).
2. Select the Delete Partition gadget.

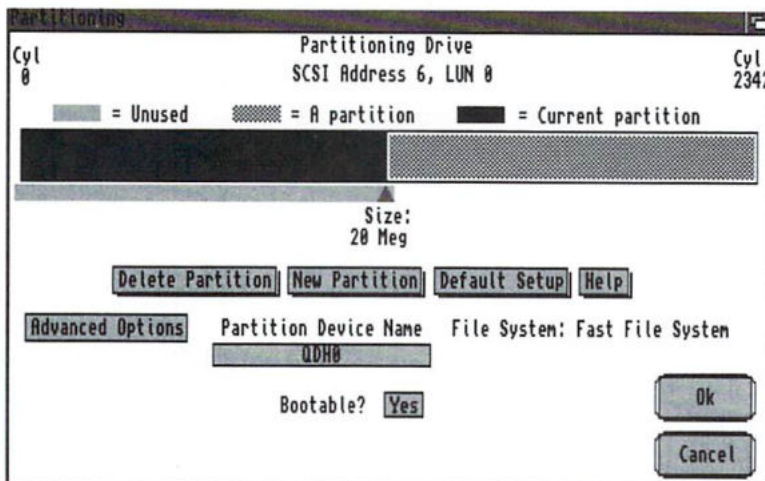
The selected partition will be deleted, and the leftmost partition in the partitioning bar will now become the selected partition.

Using HDToolbox's Default Setup for the Drive

HDToolbox contains a default setup if you decide not to form your partitions manually: a single partition on 20 MB or smaller drives, and two partitions of equal size on larger drives.

1. Select the Default Setup gadget.

The partitioning bar will form the appropriate default, for example:



The first partition will be named QDH0: and the second partition will be named QDH1:, where Q is the first letter of the name of the drive manufacturer. You may rename the partitions using the method described in "Renaming Partitions".

Saving and Formatting Your New Partitions

When you are satisfied with your new partitions, you must save them.

1. *Select the Ok gadget.*

You will be returned to the Hard Drive Preparation, Partitioning and Formatting screen.

2. *Select the Save Changes to Drive gadget.*

Your new partition information will be written to the hard disk.

3. *Wait ten seconds and reboot your system.*

An icon will appear for each partition on the hard disk. You must now format each partition to make it ready for use.

4. *Format the first new partition by clicking on the icon and selecting the Format Disk item from the Icons menu.*

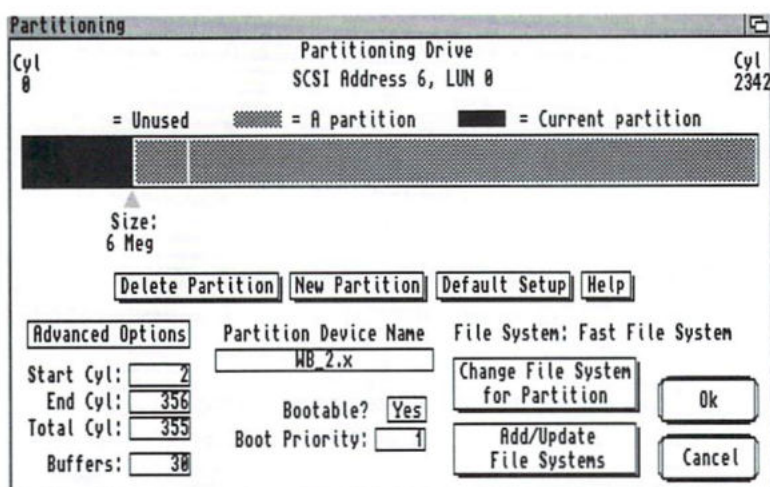
NOTE: If this is the first hard disk in the system, you will need to install Workbench and Extras on one of the partitions. See your *Introducing the Amiga* manual for complete instructions.

5. *Click on the first new partition. Select Rename from the Icons menu and change the name of the partition as desired.*

Repeat steps 4 and 5 for each new partition.

Advanced Options with Partitioning

Selecting the Advanced Options gadget on the Partitioning screen provides you with control over more detailed features of the hard disk environment.



This option was designed for advanced users who want more precise control of their partitions. Advanced Options adds more gadgets to the Partitioning screen:

(To change the following text gadgets, click in the gadget, delete the existing number and type in the new value. *Press Return after each entry.*)

Start Cyl:

Displays the number of the first cylinder of the selected partition. This number can be any cylinder in the current partition except for the last cylinder, or any cylinder in the unused area before the partition. The Total Cyl number will be adjusted accordingly.

End Cyl: Displays the number of the last cylinder of the selected partition. This number can be any cylinder in the partition except for the first cylinder or any cylinder in the unused area after the partition. The Total Cyl number will be adjusted accordingly.

Total Cyl: Displays the total number of cylinders of the selected partition. The End Cyl number will be adjusted accordingly.

Buffers: Displays the number of sector cache buffers being used in the selected partition. Buffers improve disk access time but use 512 bytes of memory per buffer. You can use as many buffers as you wish provided you have enough free memory. As a general rule, you may want to use 30 to 50 buffers for every megabyte of RAM in your system.

Bootable? Displays whether or not the selected partition can be used to boot the system. The Bootable gadget defaults to Yes for the first partition and No for all other partitions. Simply click in the gadget to toggle between Yes and No.

To use a partition as the boot partition, you will first need to prepare it by performing the following steps: Format the partition; copy your Workbench to the partition; and then reboot your system. Then you can run

	<p>HDToolbox, return to the Partitioning screen, and set Bootable? to Yes.</p>
Boot Priority:	<p>Allows you to determine which drive or partition will boot your system. This will only apply to bootable partitions. If you use a hard drive partition to boot, you should copy your Workbench into that partition.</p> <p>The value of Boot Priority can range from 127 to -128. A large value has higher priority than a lower value. The Amiga's floppy disk drive (DF0:) has a Boot Priority of 5.</p> <p>Never set a partition's boot priority above 5. It is suggested that you set your boot partition's priority to 1 and any other bootable partition to a priority of 0.</p>
Change File System for Partition	<p>Takes you to the File System Characteristics screen. This screen will allow you to change the filing system on the selected partition. (For information on this screen, see the "Modifying File Systems" section on page 6-86.)</p>
Add/Update File Systems	<p>Takes you to the File System Maintenance screen. This screen will allow you to add, delete, and modify file systems. (For information on this screen, see the "File System Maintenance" section on page 6-89.)</p>

Preparing a New Hard Disk

When you feel you need an additional large amount of storage space, you may purchase a new hard disk to mount to your existing system. Once you have physically installed your new hard disk according to the manufacturer's directions, HDToolbox is used to configure your system with all of the new hard disk's specifications.

Whenever you add a hard disk to your system, you must tell your system what type of drive you added. HDToolbox provides a list of possible drive types you may add from which you can select. Occasionally, you may add a drive type that is not on the list. In this case, HDToolbox provides a separate screen for you to enter the specifications for that particular drive. These specifications generally come with the drive you purchased and include information on the manufacturer, size, and set up of that particular drive.

NOTE: Consult your installation manual for more information on installing and using your new SCSI device. The following is a description of using the HDToolbox software and is not a complete installation guide.

If you are replacing the main drive (the one that contains Workbench), you will need to boot your system with your Install disk. Do not boot from the hard disk. Then double-click on the Install icon and follow the steps below.

To prepare your new hard drive:

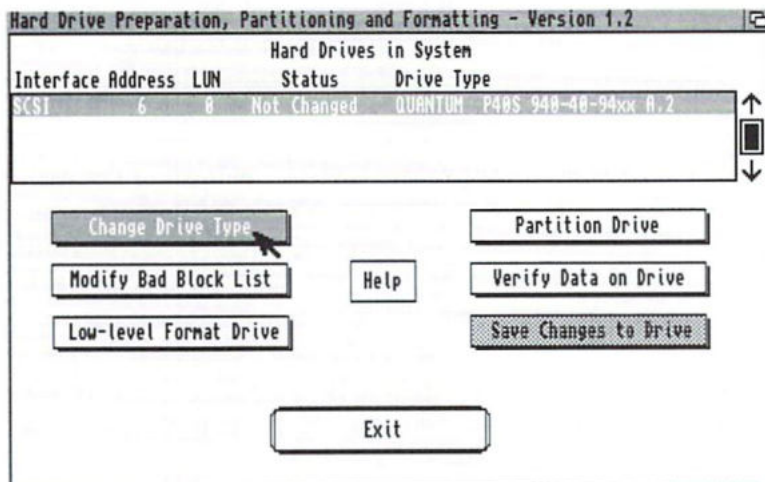
- 1. Double-click on the HDToolbox icon.**

You will see the Hard Drive Preparation, Partitioning and Formatting screen. In the scroll gadget, the new hard disk you installed is called Unknown.

2. Select the drive called *Unknown*.

This will highlight it.

3. Select the *Change Drive Type* gadget.



You are now in the Set Drive Type screen. This is where you will tell HDToolbox which type of hard disk you just added. This screen provides you with a scroll list containing a few different types of hard disks available on the market.



4. *Select the SCSI gadget or the XT gadget, depending on which type of hard disk you installed.*

NOTE: Normally, you will select SCSI. XT is only applicable to A590 users. If you select the XT gadget, a new list of XT-type drives will appear.

5. *Click on the drive type you installed if it is shown in the list.*
6. *If the appropriate drive type is not listed, select the Define New Drive Type gadget.*

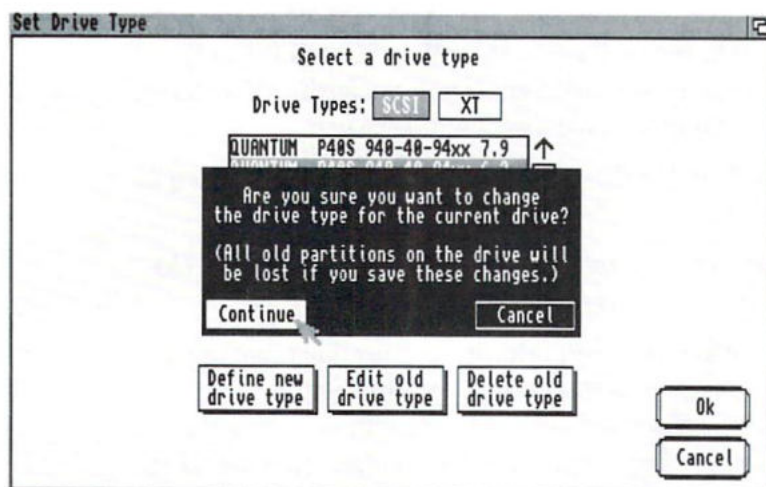
You can now type in the drive's specifications or you can have the system try to read the drive's specifications directly off the disk by selecting the Read Configuration from Drive gadget. Remember to

click on the drive type you just created. When the correct drive specifications have been read in, select the OK gadget. You will be returned to the Set Drive Type screen. The newly-defined drive type will appear on the list. Click on it before proceeding.

NOTE: You may decide to set the configuration of the drive manually, rather than having the computer read it off the drive. To do this, consult the following section "Changing the Drive Type," and then return to this section to complete the preparation.

7. Select the OK gadget on the Set Drive Type screen.

A warning requester will appear.



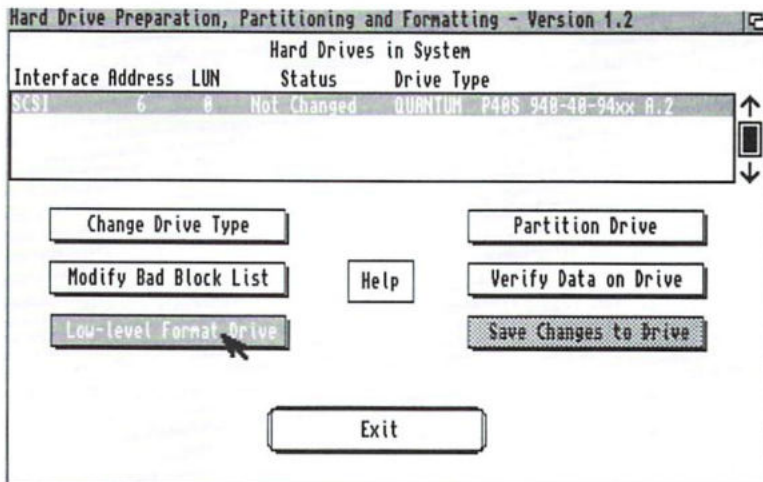
8. *Select Continue on the requester to save your changes.*

You are returned to the Hard Drive Preparation, Partitioning and Formatting screen.

The next step in hard disk preparation is low-level formatting.

Make sure that the correct drive (the one labeled "Unknown") is selected (and thus highlighted) in the scroll gadget.

9. *Select the Low-level Format Drive gadget.*



A requester will appear:



If you are installing a new drive, you don't need to worry about losing any information. Just make sure the correct drive is highlighted in the list.

You will receive a warning requester.

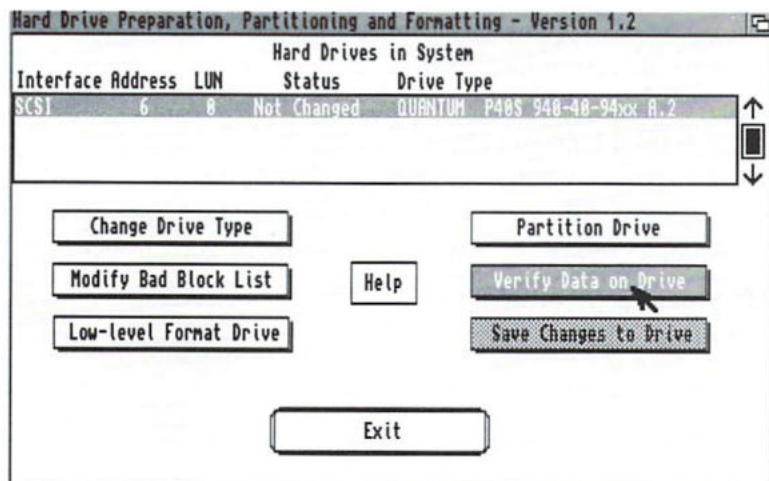
10. *Select the Low Level Format Disk gadget in the requester.*

You will receive a warning requester telling you that all information on that drive will be erased.

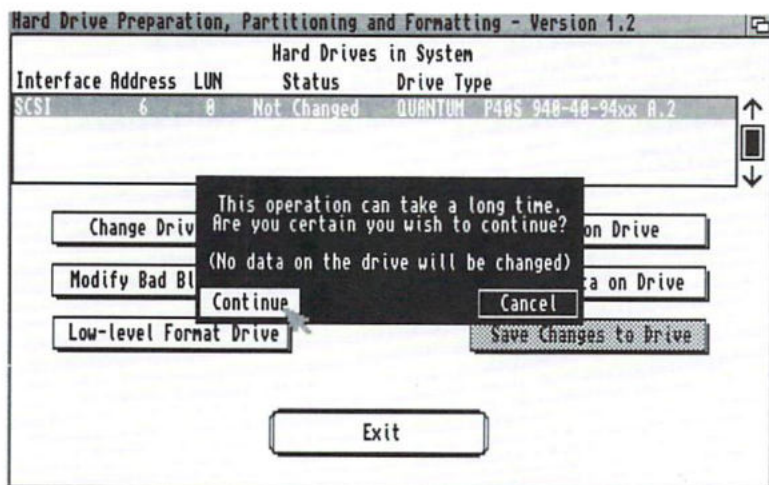
11. *Select the Continue gadget in the requester.*

The next step in preparing a new hard disk for use is verifying the disk for bad blocks.

12. Select the Verify Data on Drive gadget.



A requester will appear:



13. *Select Continue on the requester.*

If HDToolbox reports bad blocks they will automatically be recorded in the Bad Block List—a separate screen on HDToolbox. If you have received a list of bad blocks from the hard disk manufacturer, you will need to record them in a different screen on HDToolbox when you complete these preparation steps. (For information on recording bad blocks, see the section entitled “Locating Bad Blocks” on page 6-75.)

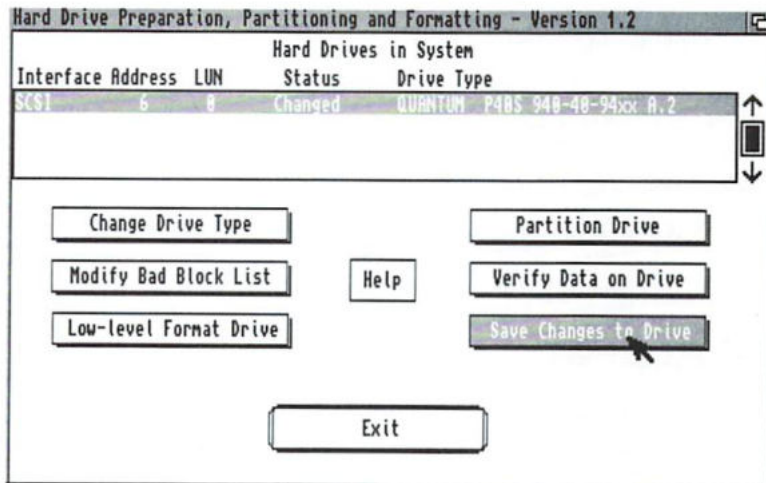
14. *Select the OK gadget.*

You will be returned to the Hard Drive Preparation, Partitioning and Formatting screen.

15. *Select the Partition Drive gadget.*

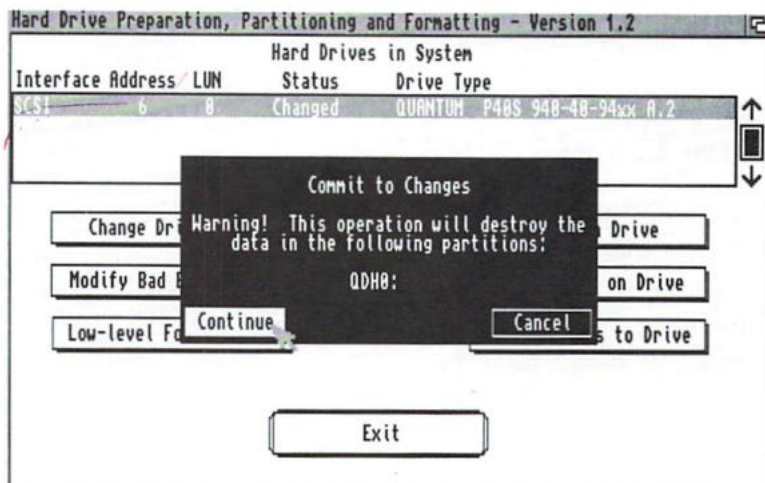
You will be taken to the Partitioning screen, where you should partition the new hard disk however you would like. For complete instructions on partitioning, see “Partitioning” on page 6-53. Then return to step 16, to complete hard drive installation.

16. *In the Hard Drive Preparation, Partitioning and Formatting screen, select the Save Changes to Drive gadget.*



This gadget saves all changes made to the hard disk configuration and overwrites the previous configuration.

You will receive a warning requester:



17. Select Continue on the requester to save your changes.



After saving a change or writing to the disk, wait at least ten seconds before turning off the power to the computer or rebooting.

18. *Wait ten seconds and reboot your system.*

An icon will appear for each partition on the hard disk.

19. *Format the first partition by clicking on its icon and selecting the Format Disk item from the Icons menu.*

NOTE: If this is the first hard disk in the system, you will need to install Workbench and Extras on one of the partitions. See your *Introducing the Amiga* manual for complete instructions.

The next step renames the partitions.

20. *Click on the first partition. Select Rename from the Icons menu and change the name of the partition as desired.*

Repeat steps 19 and 20 for each partition.

Your hard disk is now ready for use.

Low Level Formatting

After you've added a new drive, you must **low-level format** it to prepare it for operation. A low-level format must be done once to prepare the drive for use with your Amiga. This is not the same as the FORMAT command, which must also be performed on each partition.

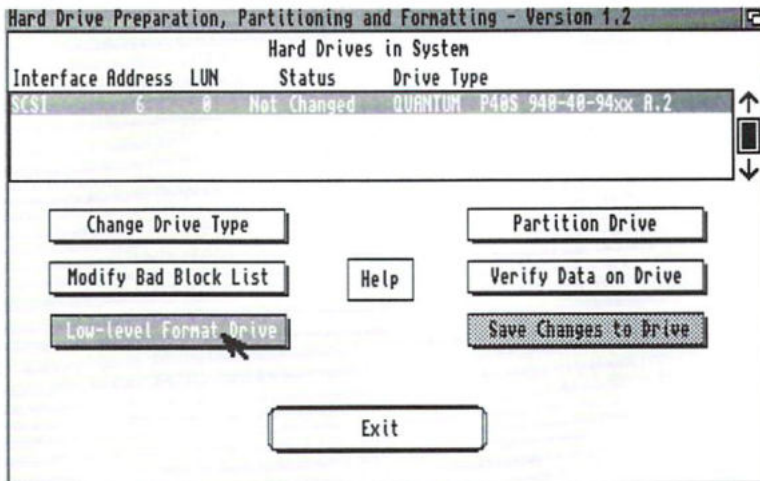
With HDToolbox, low level formatting is an easy one-step procedure that is done within the Hard Drive Preparation, Partitioning and Formatting screen.

To low level format a hard disk:

1. *Select the drive you wish to low level format by clicking on it in the scroll area.*

The selecting drive will be highlighted.

2. *Select the Low-level Format Drive gadget.*



A requester will appear:



Select the Low Level Format Disk gadget in the requester.

At this point, you will receive a warning requester telling you that all information on that drive will be erased.

3. *Select the Continue gadget on the requester.*

This step may take as little as a few seconds or as much as several minutes, depending on the type of hard drive you are using.

4. *Select the Save Changes to Drive gadget.*

The low-level format is complete.

Locating Bad Blocks

A **bad block** is a portion of the hard disk which can no longer be read. Just as floppy disks can develop errors and corruption from being used over and over again, hard disks can also develop errors. Hard disk errors, however, occur much less frequently.

If you consistently find read/write errors on backups, it may be because of bad blocks. Other symptoms of bad blocks include frequent hardware and software failures and requesters. If you've added a new hard disk and low level formatted it, you should locate bad blocks before you begin to enter data.

With HDToolbox you can easily check your hard disk(s) for errors. The program will search your hard disk(s) and report a list of blocks which have developed errors. These locations are then recorded on a separate screen known as the Bad Block screen.

The computer will use the recorded blocks on the Bad Block List (located on a separate screen) during initialization to avoid using these areas. If HDToolbox finds errors in areas which contain data, it will re-write the data to a different area on the hard disk, if possible.

It is not unusual for a new hard disk to have a few bad blocks before it is even used. Often, the company which made the hard disk will provide you with a list of bad blocks. You must enter these locations in the Bad Block list.

Just as regular backups are necessary for proper hard disk maintenance, occasional data verification is also necessary. Depending on how often you use your hard disk, you may want to check the integrity of your hard disk from as often as once a week to once a month.

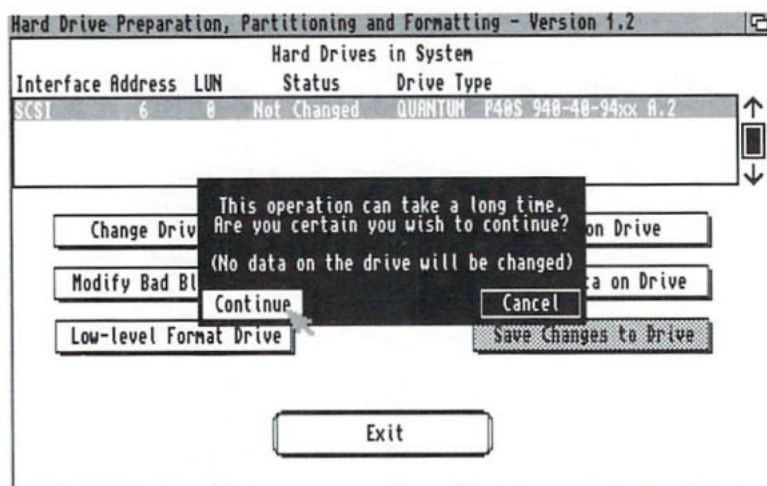
To check a hard disk for bad blocks, start from the Hard Disk Preparation, Partitioning and Formatting screen:

1. *Select the hard disk you wish to verify by clicking on its name in the scrolling list.*

The selected drive will be highlighted.

2. *Select the Verify Data on Drive gadget.*

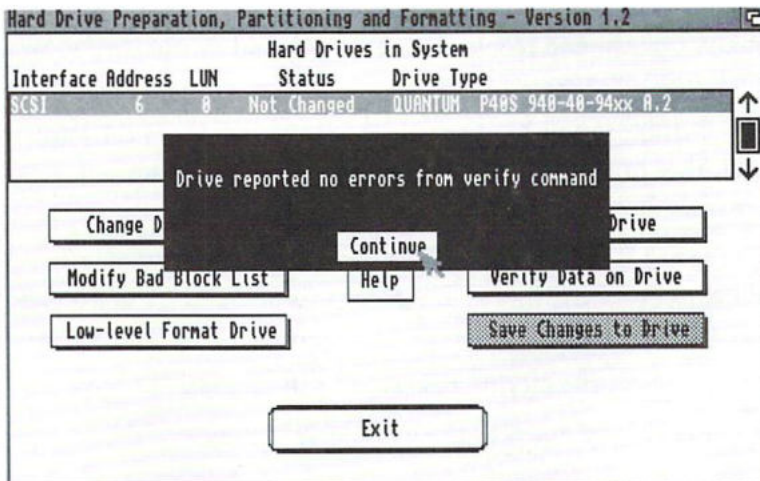
You will receive a warning requester.



Verifying the data on a typical 40MB hard disk can take as little as a few seconds or as much as several minutes, depending on the type of hard drive you are using.

3. Select *Continue* to begin the process.

If HDToolbox finds no errors, you will receive the following requester:



4. Select *Continue* and the verify is complete.

If HDToolbox finds errors, you will receive a requester listing the bad blocks with their locations on the hard disk. These bad blocks will be recorded on the Bad Block List. To view this list, or to add bad blocks supplied to you by the manufacturer, read the following section, "Adding a Bad Block to the Bad Block List".

Adding a Bad Block to the Bad Block List

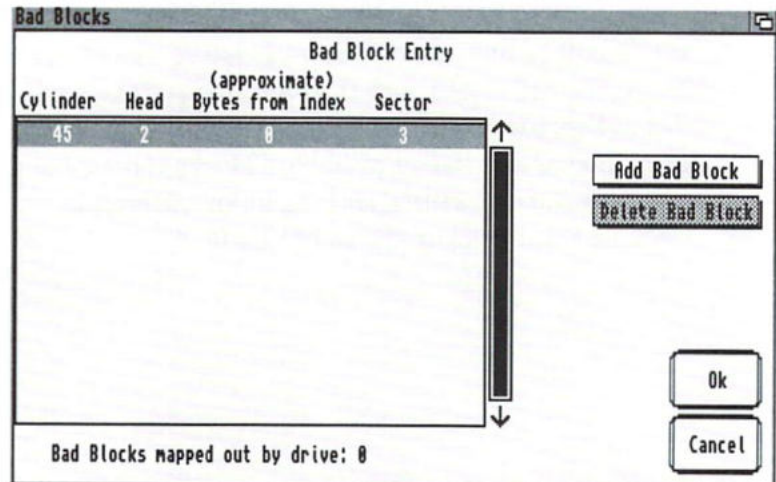
If the hard disk manufacturer has provided you with a list of bad blocks you must enter them into HDToolbox on the Bad Blocks screen. You may also choose to look at this screen to make sure the bad blocks which were found by the Verify Data on Drive function were recorded.

The Bad Blocks screen keeps a list of any blocks on the hard disk that might develop read/write errors. The computer uses this list to avoid using these areas.

To see the Bad Blocks List, start from the Hard Drive Preparation, Partitioning and Formatting screen. To add a bad block to the list:

1. *Select the Modify Bad Block List gadget.*

You will be taken to the Bad Blocks screen.



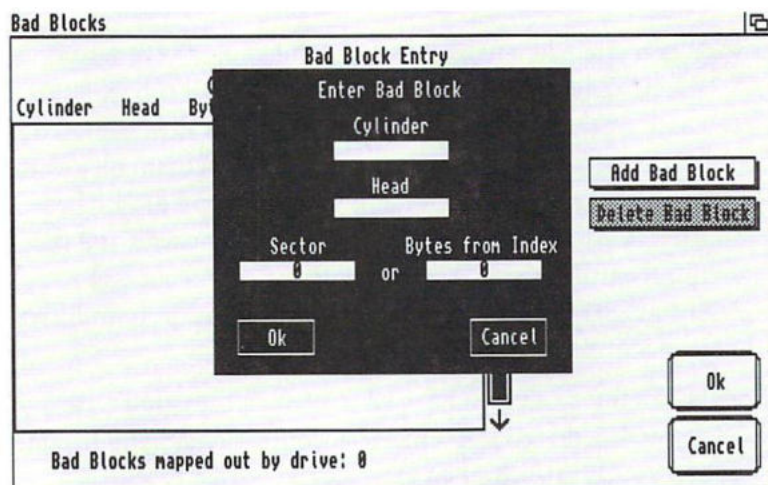
Notice the scroll gadget in the center of the screen. If you have not previously located any bad areas on your hard disk, the gadget will be empty. If bad blocks were found, they will be listed here.

The list shows the location of the bad blocks by cylinder, head, bytes from index and sector. Note that the system will list a range for the approximate number of Bytes from Index, and you will only be able to list one error per sector. Once a sector has an error, the entire sector will be marked as bad.

Most SCSI hard disks handle bad block errors internally—they locate bad blocks themselves and avoid using these areas without any user intervention. When you use the Verify Data on Drive function, it locates bad blocks which the hard disk itself did not find. At the bottom of the screen is an informational message called Bad Blocks mapped out by drive: which displays the number of bad blocks the hard disk located internally.

2. Select the Add Bad Block gadget.

A requester will open.



In this requester, enter the Cylinder, Head, Bytes from Index or Sector of the block. To enter this information, click on each text gadget and enter the proper number. Press Return after each entry. Select OK to add this block to the list or Cancel. You will be returned to the Bad Blocks screen.

To delete a bad block from the list, click on it and select the Delete Bad Block gadget.

3. *Select OK on the Bad Blocks screen to save your changes.*

You will be returned to the Hard Disk Preparation, Partitioning and Formatting screen.

4. *Select the Save Changes to Drive gadget.*

Changing the Drive Type

In order for your system to function properly with a hard disk, it must be told which type of hard disk it is using. The configuration of the hard disk that came with your computer is already set. You will not need to change anything.

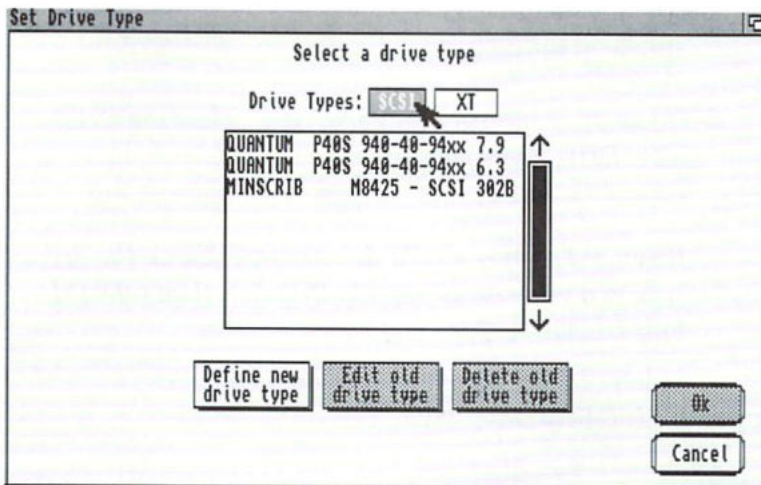
When you add a new hard disk, you supply your system with the hard disk's specifications through HDToolbox. HDToolbox can read the specifications directly from the new hard disk. You may, however, decide to type them in manually from the documentation that came with the hard disk. These specifications include the manufacturer's name, the model number, the storage size, and many other technical specifications. The use of Read Configuration from Drive is recommended.

HDToolbox can also be used to change the drive type if you've replaced a drive, or to change specifications if you need to edit any specifications of an existing hard drive type.

To change or add a new drive type, start from the Hard Drive Preparation, Partitioning and Formatting screen:

1. Select the Change Drive Type gadget.

This gadget takes you to the Set Drive Type screen.



This screen lists the types of drives whose specifications are stored on the disk. A sample list of drives which you might add to your computer is provided for your convenience. If the drive type you've added is not on the list proceed to the following section, "Defining a New Drive Type".

2. If you've added a drive type which is on the list, select the correct drive then select OK.

You will be returned to the Hard Drive Preparation, Partitioning and Formatting screen.

3. Select the Save Changes to Drive gadget.

4. Wait ten seconds and reboot.

Editing a Drive Type or Defining a New Drive Type

Start from the Hard Drive Partitioning and Formatting screen:

1. *Select the Change Drive Type gadget.*

This gadget takes you to the Set Drive Type screen. This screen lists the types of drives whose specifications are stored on the disk.

2. *If you are adding a new drive type, select the Define New Drive Type gadget.*

OR

If you are editing an old drive type, click on the drive type you will edit in the scrolling list, then select the Edit Old Drive Type gadget.

Either of these selections will take you to the Define/Edit Drive Type screen.

Define/Edit Drive Type

Define a New Drive Type

Filename:

Manufacturers name:

Drive Name:

Drive Revision:

Cylinders:

Heads: Size: 21046K (20 Meg)

Blocks per Track:

Blocks per Cylinder:

Supports reselection? ☒ Yes

Park head where (cylinder):

Reduced Write Current

Cylinder:

Write Precomp

Cylinder:

NOTE: When creating a new drive type with the same name as an existing drive type, the computer will only use the version with the most recent date. In order to save the correct change, make sure that your system clock shows the current date and time.

If you wish, you may select the Read Configuration from Drive gadget and HDToolbox will automatically record the drive specifications. Then select the OK gadget to return to the main screen.

If you wish to record the specifications manually, proceed with the following steps. Use the specifications supplied by your hard drive manufacturer to enter the required information explained below.

When you are entering the specifications of a new drive type you will need to click on the appropriate text gadget, delete the existing information, type the correct information, and press Return. *Always press Return after typing in new information.*

The specifications on the Define/Edit Drive Type screen are as follows:

Filename:	The file called drive definitions is located on your hard disk and contains all of the drive specifications you have saved. This is the list of sample drives you saw on the Change Drive Type screen. Since you can save multiple drive types and their specifications in this file, you do not need to change this filename.
Manufacturer's name:	Displays the name of the drive manufacturer, using up to eight characters.

Drive Name:	Displays the name of the drive, using up to sixteen characters.
Drive Revision:	Displays the number of the drive revision, using up to four characters.
Cylinders:	Displays the number of drive cylinders.
Heads:	Displays the number of drive heads.
Blocks per Track:	Displays the number of blocks (512 bytes per block) on each track. Some manufacturers may list this as "sectors".
Blocks per Cylinder:	Displays the number of blocks in each cylinder. This will normally be the number of heads multiplied by the number of blocks per track.
Size:	<p>Displays the amount of memory space on the drive in kilobytes (K) or megabytes (MB).</p> <p>After you have entered information into the Cylinders:, Heads:, and Blocks per Track: text gadgets and pressed Return, the value listed after Size will change. When you are finished, compare the listed size to the drive specification, to ensure that it is close to the value given by the drive manufacturer.</p>
Reduced Write Current Cylinder:	Not used with SCSI devices.
Write Precomp Cylinder:	Not used with SCSI devices.

Supports reselection?	Refer to the manufacturer's documentation to determine whether or not a SCSI device supports reselection. Click on this gadget to change it.
Park head where (cylinder):	Displays the number of the cylinder recommended by the manufacturer. This function is not needed with drives that automatically park the drive head. Refer to the manufacturer's documentation. If no value is given by the manufacturer, use the number of the last cylinder.
Ok	Saves the changes to this screen and returns you to the Change Drive Type screen.
Cancel	Returns you to the Change Drive Type screen without saving your changes

When you have finished entering the specifications:

1. **Select the Ok gadget on the Define/Edit Drive Type screen.**

You will be returned to the Set Drive Type screen.

2. **Click on the newly defined drive in the drive type list.**
3. **Select the Ok gadget on the Set Drive Type screen.**

This saves your changes and returns you to the Hard Drive Preparation, Partitioning and Formatting screen.

4. **Select the Save Changes to Drive gadget.**
5. **Wait ten seconds and reboot.**

Modifying File Systems

NOTE: This function is intended for advanced users.

A **file system** is software that controls how data is organized on a disk. Amiga systems use the FastFileSystem (FFS), which is an efficient file system saving time and hard disk space.

You may decide to switch to a different filing system—perhaps to an upgrade in AmigaDOS or to a file system you’ve produced yourself. HDToolbox allows you to modify the list of available filing systems by adding new file systems, deleting file systems, and modifying existing file systems.



Changing the filesystem of a partition that contains data is likely to make the data inaccessible.

Advanced users may choose to use the File System Characteristics screen to modify a partition’s file system. Most users can safely ignore this screen.

To modify a partition’s file system, start from the Hard Disk Preparation, Partitioning and Formatting screen:

- 1. Select the Partition Drive gadget.***

The Partitioning screen will be displayed.

- 2. Select the Advanced Options gadget.***

The Advanced Options screen will be displayed. You must next make the partition whose file system you wish to modify the current partition. To do this, click on that partition in the partitioning bar.

3. Select the Change File System for Partition gadget.

The File System Characteristics screen will be displayed. At the top of the screen, it shows the name of the selected partition.

The screenshot shows a window titled "File System Characteristics" with a subtitle "Partition WB_2.x". Inside the window, there are four buttons arranged in a 2x2 grid: "Fast File System" (highlighted), "Old File System", "Custom File System", and "Reserved Partition". Below these buttons, there is a checkbox labeled "Automount this partition?" with "Yes" selected. Further down, there are several input fields: "Identifier =" with the value "0x444538f", "Mask =" with "0xffffffff", "MaxTransfer =" with "0xffffffff", "Reserved blocks at beginning:" with "2", and "end:" with "0". At the bottom, there are two more checkboxes: "Use custom boot code?" with "No" selected, and "Number of custom boot blocks?" with "0". On the right side of the dialog, there are "Ok" and "Cancel" buttons.

You must next choose the file system for the selected partition by clicking on one of the following gadgets:

Fast File System

This is the default filing system.

Old File System

This filing system trades speed for recoverability. If part of the disk were to become unreadable, it might be easier to recover information stored with this filing system.

Custom File System	This allows you to install your own filing system.
Reserved Partition	This will allow you to reserve an area on the disk without a partition. This area can be set aside for some special use, such as for a UNIX operating system.

After you have chosen a file system you must set the values for the following:

(To change any of the following values, click on the box, delete the existing information, type the new number and press Return. Hex numbers must begin with 0x.)

Identifier =	Displays the hex number (code) that tells AmigaDOS what filing system is being used. The Identifier can only be modified when using a Custom File System.
Mask =	Displays the hex number that defines which areas of memory can be used with Direct Memory Access (DMA). Mask is available when using Fast File System and Custom File System.
MaxTransfer =	Displays the hex number that determines the maximum number of bytes to be moved during each DMA transfer. MaxTransfer is only available when using Fast File System and Custom File System.

Reserved blocks
at beginning:

Displays the number of blocks reserved at the beginning of the selected partition for DOS usage. This value defaults to 2, and normally should not be set to less than 2.

Reserved blocks
at end:

Displays the number of blocks reserved at the end of the selected partition, for DOS usage. This value defaults to 0.

To return to the Partitioning screen without saving your changes, select the Cancel gadget.

To save your new file system characteristics:

1. *Select the Ok gadget.*

You will be returned to the Partitioning screen.

2. *Select the Ok gadget in the Partitioning screen.*

You will be returned to the Hard Drive Preparation, Partitioning and Formatting screen.

3. *Select the Save Changes to Drive gadget.*

File System Maintenance

The File System Maintenance screen allows you to modify the list of available filing systems. This section tells you how to:

- Add a new file system
- Delete a file system
- Modify an existing file system

To use this screen to perform any of the above, you must start from the Hard Disk Preparation, Partitioning and Formatting screen:

1. Select the Partitioning gadget.

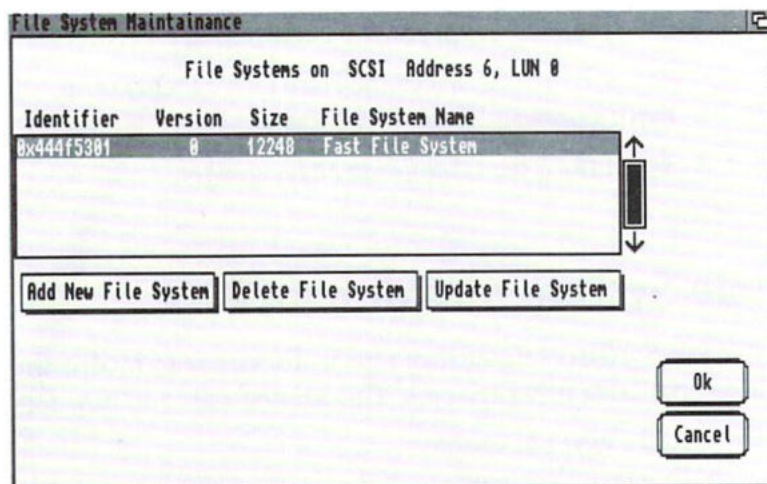
The Partitioning screen will be displayed.

2. Select the Advanced Options gadget.

The Advanced Options will be displayed. You must next make the partition whose file system you wish to modify the current partition. To do this, click on that partition in the partitioning bar.

3. Select the Add/Update File System for Partition gadget.

You will be taken to the File System Maintenance screen.



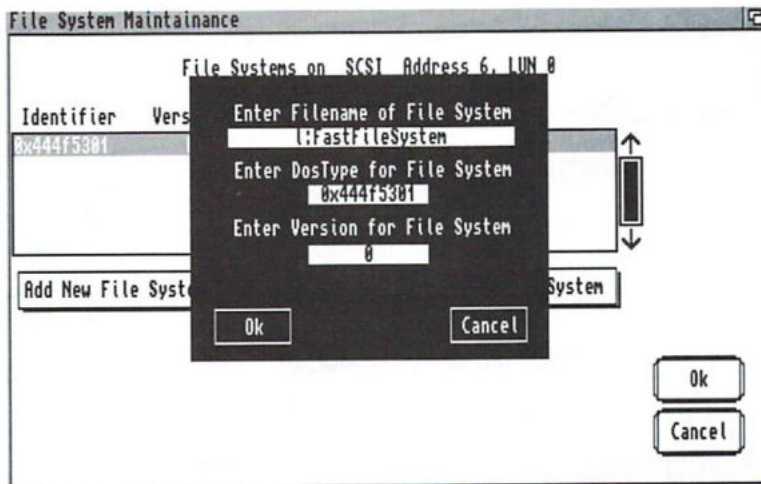
At the top of the screen, the selected drive is displayed by address and LUN. Below, in the scrolling list each file system stored on that drive is displayed, showing its Identifier hex number, Version number, Size in bytes, and File System Name.

To Add a New File System

Start from the File System Maintenance screen:

1. *Select the Add New File System gadget.*

A small window will appear.



2. *Delete the contents of the first text gadget and type in the full pathname to the location of the new file system. Press Return.*
3. *Click on the second text gadget, delete the existing hex number and type the hex number of the DosType of the new file system.*

The system defaults to FastFileSystem with DosType 0x44f5301.

4. *Click on the third text gadget and type in the version number of the new file system.*

5. *Select the Ok gadget to retain your changes.*

You will be returned to the File System Maintenance screen.

6. *Select the Ok gadget in the File System Maintenance screen.*

You will be returned to the Partitioning screen.

7. *Select the Ok gadget in the Partitioning screen.*

You will be returned to the Hard Drive Preparation, Partitioning and Formatting screen.

8. *Select the Save Changes to Drive gadget.*

To Delete a File System

Start from the File System Maintenance screen:

1. *Select the file system you wish to delete.*

The selected file system is highlighted. To select a different file system, click on the desired file system.

2. *Select the Delete File System gadget.*

The selected file system will disappear.

3. *Select the Ok gadget in the File System Maintenance screen.*

You will be returned to the Partitioning screen.

4. *Select the Ok gadget in the Partitioning screen.*

You will be returned to the Hard Drive Preparation, Partitioning and Formatting screen.

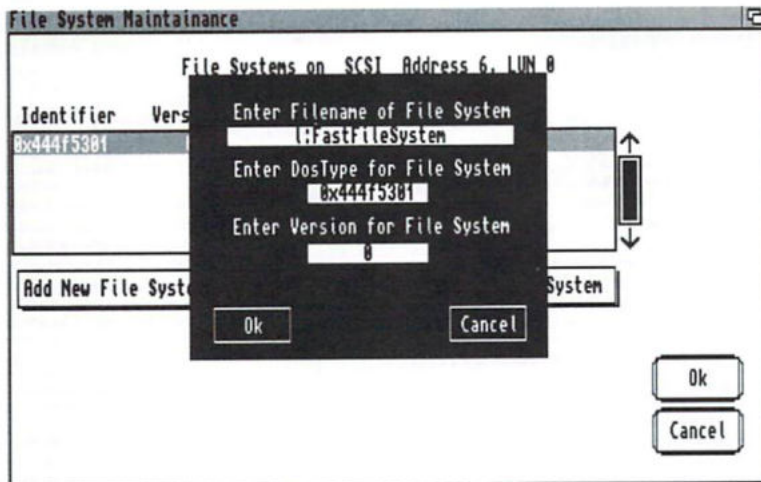
5. *Select the Save Changes to Drive gadget.*

To Update an Existing File System

Start from the File System Maintenance screen:

1. *Select the Update File System gadget.*

A small window will appear.



2. *Delete the contents of the boxes which must change and type the new information. Press Return.*

(See "To Add a New File System" on page 6-91 for more information on these specifications.)

3. *Select the Ok gadget to retain your changes.*

You will be returned to the File System Maintenance screen.

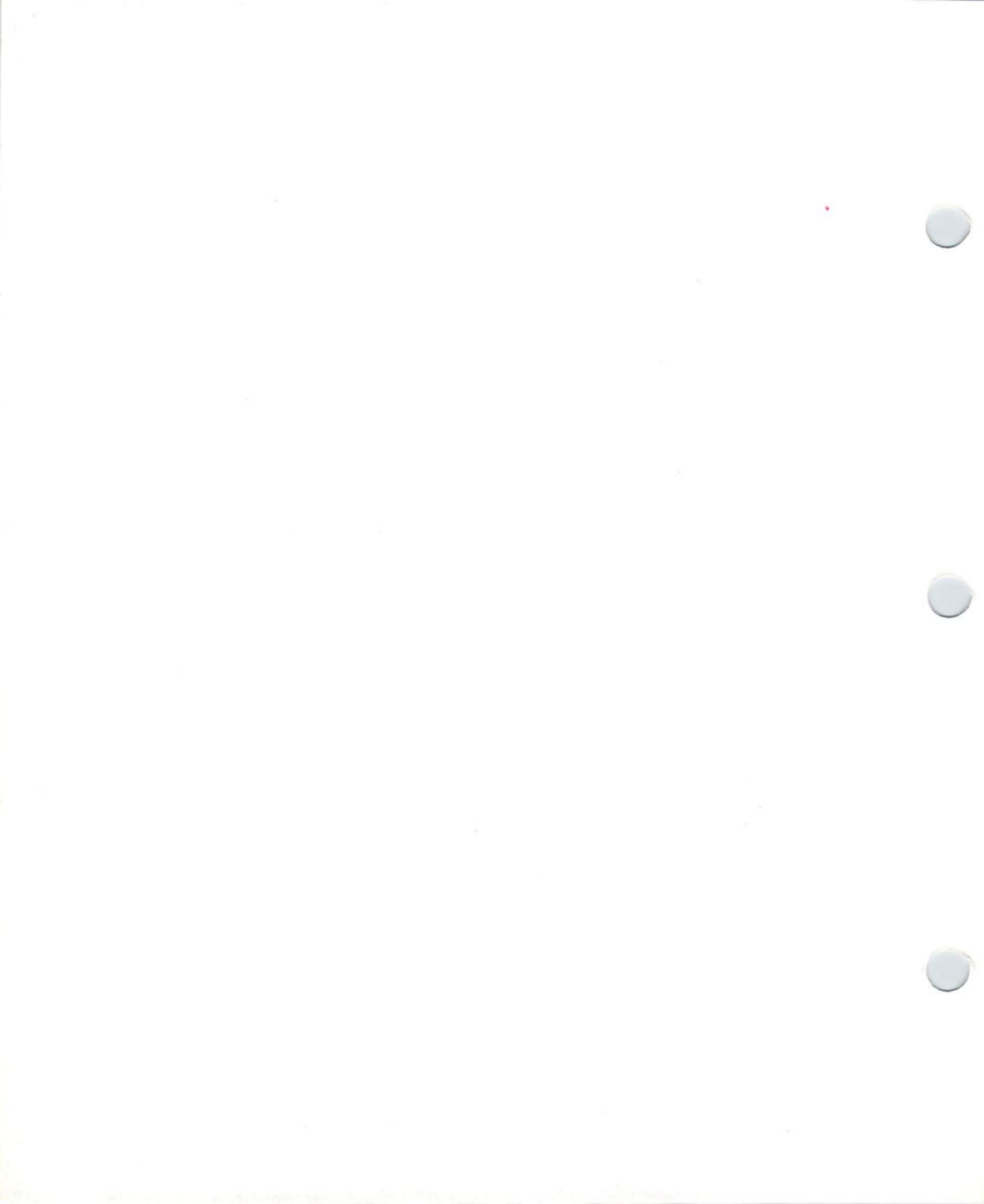
4. *Select the Ok gadget in the File System Maintenance screen.*

You will be returned to the Partitioning screen.

5. *Select the Ok gadget in the Partitioning screen.*

You will be returned to the Hard Drive Preparation, Partitioning and Formatting screen.

6. *Select the Save Changes to Drive gadget.*



Chapter 7. Using AmigaDOS

This chapter introduces you to the basic concepts of AmigaDOS, such as

- the hierarchical file system
- how to use the Shell and some basic AmigaDOS commands
- the features of the Shell
- how to run programs through the Shell
- how to use and edit scripts, including the Startup-sequence
- how to make the most of your system if you only have one floppy drive

For full information on the AmigaDOS commands, see Chapter 8, "AmigaDOS Reference."

Introduction to AmigaDOS

AmigaDOS is the Amiga Disk Operating System. A **disk operating system** is software that controls the basic functions of the Amiga, such as:

- providing a filing system which organizes the data (information) programs use and produce.
- handling the storage and retrieval of information from floppy and hard disks.
- letting you run more than one program at a time (multitasking).
- providing an interface to peripheral devices like printers and modems.

Even when you are using application programs like word processors, paint packages, or music programs, AmigaDOS is in control, making sure everything is available when you need it.

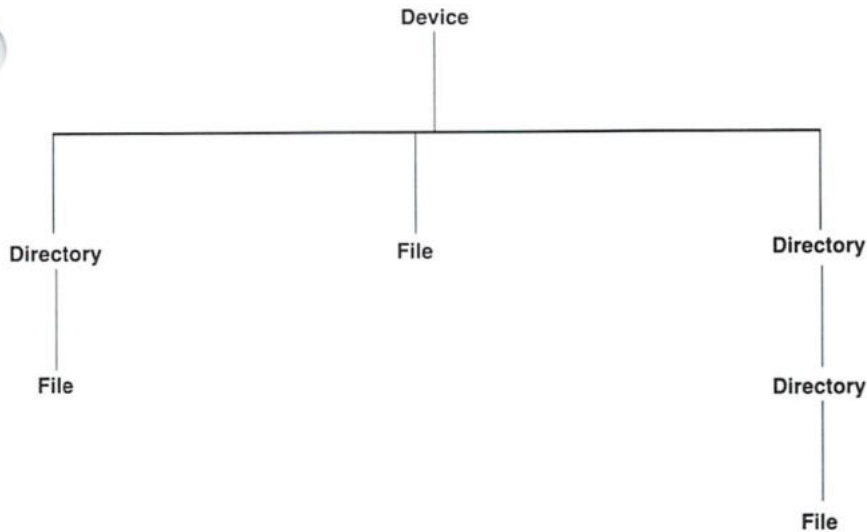
You can communicate with the Amiga through typed AmigaDOS **commands**. Some of these commands parallel Workbench operations, such as COPY, RENAME, and FORMAT. There are also advanced commands that allow you to create scripts (text files) for performing repetitive tasks, repair corrupt disks, monitor the use of memory, and perform other tasks unavailable through the Workbench.

The commands are entered through a special window, known as a Shell window. You can start a Shell by opening the Shell icon in the Workbench2.0 disk window or by entering the NEWSHELL command, either from within a Shell window or in the Execute Command requester.

Before you can enter commands, you need to understand how AmigaDOS stores information and how to reference the files and devices used by AmigaDOS. This is explained in the following section, "The File System."

The File System

AmigaDOS stores information on devices which are organized according to a system of directories, subdirectories, and files. This is known as a **file system**. Directories and files are arranged in a hierarchical system often referred to as a tree since one way to sketch out a directory structure is with a branching diagram that looks like a family tree. The branches are directories, any of which can include other directories. At the ends of the branches are the files.



The different components of the file system are explained in this section.

Devices

You can store information on floppy disks, hard disks, or magnetic tape. To gain access to a file on a particular disk, you can refer to the disk by its **volume name**. The volume name is a name given to a disk, such as Workbench2.0, Extras2.0, or Empty. When you refer to a disk by volume name, the system will search all the available drives for the disk. If it cannot find a disk of that name, a requester will appear asking you to insert the disk. If there is more than one volume with the same name, you will have to refer to them by device names, or the system will randomly choose which disk to access.

Another way to refer to disks is by **device name**. A device name refers to a particular device, such as a floppy disk drive or hard disk partition. For instance, DF0: is the device name of the Amiga's internal disk drive. If you save a file to DF0:, it will be saved on whichever disk is inserted in DF0: at that time.

The Ram Disk represents another type of device, RAM:. The RAM: device is a portion of the Amiga's internal memory that can be used as a storage device. However, all information in RAM: is lost if the Amiga is rebooted or powered off.

Device and volume names must be followed by a colon (:).

Peripheral Devices

Peripheral devices can also be accessed through AmigaDOS. AmigaDOS has assigned standard names for devices that are attached to the various ports as well as to the windows that appear on the screen. These devices are typically used for output, such as when you want to copy a file to the printer or send information through a modem. The standard device names are listed below:

- PAR: Represents any device, usually a printer, that is connected to the parallel port. If you copy a file to PAR:, it will be sent to the device attached to the parallel port.
- SER: Represents any device connected to the serial port, such as a printer or a modem.
- PRT: Represents the printer selected with the Printer editor in the Prefs drawer.
- CON: Represents a console, which uses a window to accept typed input and display output. The Shell window is one kind of console window.
- Represents the current window.

NIL: Represents a dummy device commonly used to prevent output from appearing on the screen. All output sent to NIL: is discarded.

Directories

Directories allow you to group and classify related files together, making it easier to find and work with them. For example, you can use two different directories to separate letters from spreadsheets or to keep files belonging to one person distinct from those belonging to another.

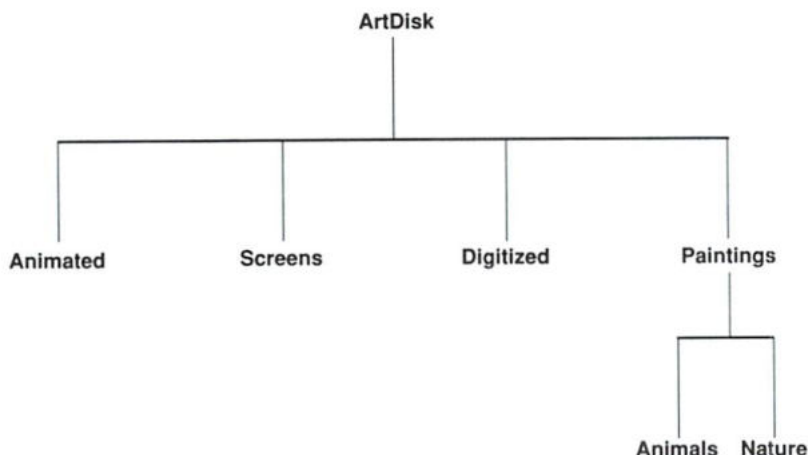
Each file on a disk is located in a directory. An empty, formatted disk contains one directory, the **root directory**. If you create a file on an empty disk, that file resides in the root directory. (If the file had an icon attached to it, the icon would appear in the disk window.) Any other directories you create reside in the root directory.

Directories can contain files as well as other directories, called **subdirectories**. Subdirectories are like drawers within drawers. They are just one more way of organizing information.

For instance, suppose you had a disk named ArtDisk and you needed to store animated files, screen shots, digitized art, and paint files. You could create four directories: Animated, Screens, Digitized, and Paintings. If you have several paint files that are of animals and several that are of nature scenes, you

Directories are the AmigaDOS version of the Workbench drawers.

could create two subdirectories within the Paintings directory: Animals and Nature. Your disk organization might look like this:



The whole purpose of the directory structure is to organize your files.

Files

A **file**, the basic unit of storage on a computer, is an organized collection of information referred to by a name. All the programs you run and any permanent data that a program uses or produces are files.

On the Workbench, files are represented by tool and project icons. Tool icons represent program files. This includes application programs, such as word processors or CAD programs, as well as the AmigaDOS commands. Project icons represent data files. Data files contain the information created or used by a program, such as text, graphic, and spreadsheet files.

.info Files

Another type of file used by the Amiga is a .info file (pronounced "dot info"). The .info files correspond to the icons that appear on the Workbench screen. Every file or directory that has an icon also has a corresponding .info file. For instance, the Workbench2.0 disk contains two files for the Clock program: one called Clock which contains the Clock program, and another called Clock.info which contains the data needed to display the Clock icon.

In addition to storing the graphics and position data for the icon image, a .info file also contains any Default Tool or Tool Type information entered into the icon's Information window.

When you are working through the Shell, AmigaDOS does not automatically associate .info files with the corresponding files or directories. For instance, if you copy the Clock file from the Utilities directory to the System directory (using the COPY command), the Clock.info file will not be copied the way it would if you had dragged the icon from one drawer to another. If you want the Clock icon to appear in the System drawer, you have to copy the Clock.info file as well.

If you copy .info files in order to change the icon images, be sure you copy an icon of the same type as the item it represents: Tool, Project, Drawer, Disk or Trashcan. If the icon's type does not match the type of file it represents, you may have problems when you try to open the icon from the Workbench. (An icon's type is displayed in the icon's Information window. You can change an icon's type with the IconEdit program.)

NOTE: You cannot delete disk icons. (A disk.info file corresponds to a disk icon.) If you delete the disk.info file, a default disk icon will automatically replace the previous icon.

Paths

To specify a file (for example, to copy it or rename it) you must specify the complete **path** that leads to that file along with the name of the file. The path includes:

- The volume or device name. A volume name refers to a specific disk, such as Workbench2.0:. A device name is used as a general reference to the disk in a device at that time, such as DF0:.
- All directories and subdirectories that contain the file.
- The filename.

For instance, the path to the file Readme in subdirectory Fractals, directory Demos, on the disk called MyDisk in drive DF0: would be:

DF0:Demos/Fractals/Readme

or

MyDisk:Demos/Fractals/Readme

The path starts at the volume level and moves down. The first element of the path is always the device or volume name (to the left of the colon), followed by the root directory (the colon), then the names of any directories and subdirectories (separated by slashes), and finally the filename. This gives the exact location of the file.

Naming Files and Directories

File and directory names can be up to 30 characters long and can contain capital letters and any punctuation marks that do not have special meanings. You cannot use a colon (:), semicolon (;), asterisk (*), slash (/), question mark (?), back apostrophe ('), number or pound sign (#), or a percent sign (%). (These special characters are explained on page 7-29).

Any capitalization you use in a filename will be preserved, but AmigaDOS is case indifferent. This means that it will recognize the name by the characters, not by the case — TextFile is the same as textfile.

While you can insert spaces in names, you should avoid them when working through AmigaDOS. When using names with spaces, the entire path containing the name must be enclosed in double quotes. For instance, you must type:

```
"DF0:Text File"
```

It is generally better to use an underscore (_) or period (.) as a separator. Be especially careful not to put a space at the beginning or end of a name. The space will be invisible when the name is displayed, but AmigaDOS will not recognize the name without the space included.

If there is a conflict between a name and a command **keyword**, enclosing the name in quotes will ensure that it is interpreted correctly. For instance, if you have a directory named Files, you would have a conflict with the LIST FILES command which lists only the files in a directory excluding subdirectories. To get around this, you would have to type:

```
LIST "Files"
```

You can have files with the same name so long as they are in different directories.

A keyword is a special word recognized by an AmigaDOS command to identify an argument or specify an option

Quick Review

- A device refers to a physical device like a disk drive or printer or a software device, like the Ram Disk or a window.
- A volume is another term for a floppy disk or hard disk partition.
- You can refer to disks by volume name, such as MyDisk:, or a device name, such as DF0:. They can be used interchangeably, but with either you must always include the colon (:) at the end.
- Directories are equivalent to Workbench drawers.
- The root directory is at the top of the filing system for a given disk—the one that contains all the other directories.
- A subdirectory is a directory that is contained within another directory (a drawer within a drawer.)
- Files are named collections of data.
- A path is the series of device, directory and subdirectory names used to reach a particular file.

Basic AmigaDOS Commands

This section provides an introduction to using the Shell and some of the most basic AmigaDOS commands. It is not meant to give full instructions on each and every command, but it should help you become familiar with the basic capabilities of AmigaDOS.

Types of Commands

There are two types of AmigaDOS commands: disk-based and **Internal**.

Every time a disk-based command is invoked, the command must be loaded from the disk before it can be executed. For hard disk users, this is a fast process as the commands are always accessible to the system. However, for floppy disk users, the command must be read from the Workbench disk used to boot the system. If the disk is not currently in the disk drive, it must be inserted so that the command can be read.

The Internal commands are built into the Shell, which is in ROM. When an Internal command is invoked, the system can access it immediately. It does not have to be read from disk.

Many of the AmigaDOS commands parallel menu items or programs on the Workbench. However, it is often quicker and more convenient to perform these actions through typed commands. Some people simply prefer typing to using the mouse.

The commands included in this section and their Workbench counterparts, if any, are shown below:

Basic AmigaDOS Commands		
Command	Function	Workbench Counterpart
DIR	Show files on disk	Show All Files
LIST	Show files, with sizes, etc.	View By Name
INFO	Show information on all disks	Title bars
MAKEDIR	Make a new directory	New Drawer
COPY	Copy a file or directory	Copy
CD	Change the current directory	Select another window
PATH	Add a directory to the search path	
TYPE	Display the contents of a file	More
RENAME	Rename a file or directory	Rename
DELETE	Delete a file or directory	Delete
DATE	Set the correct date and time	Time editor
SETCLOCK	Save the date and time	Time editor
FORMAT	Format a disk	Format Disk
RELABEL	Rename a disk	Rename
DISKCOPY	Copy a disk	Copy
NEWSHELL	Open a new Shell window	Open Shell icon
ENDSHELL	Close a Shell window	Select Shell window close gadget

If you have a hard disk, you may want to reboot your machine with the Workbench2.0 disk so that the information shown on your screen matches the information given in this section. If you proceed from hard disk, you may see slightly different output displayed on your screen, but the commands will work in essentially the same way.

All the AmigaDOS commands are shown in capital letters to distinguish them from the rest of the text. It is not necessary to type them in capital letters, as the Amiga ignores case differences in commands or filenames.

In the following examples, if the command results in output by the Amiga, that output is shown on the line underneath the command.

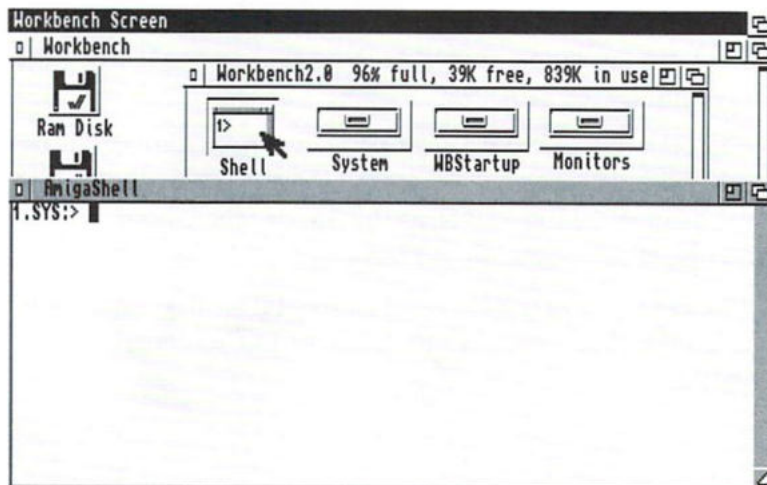


The Shell

You communicate with AmigaDOS through a Shell, a special window which accepts text input. Shell windows can be dragged, resized, overlapped, and zoomed just like other Workbench windows. However, you cannot drag icons into the Shell or use the mouse to perform any Workbench-style operations in it (except for cutting and pasting which is described in the "Features of the Shell" section).

You enter AmigaDOS commands at a text **prompt**, usually ending in a > symbol. After typing in the command and any other necessary information, such as filenames or command options, press Return. The command is then executed by the Amiga. The information typed after the prompt up until you press Return is known as the **command line**.

1. *Open a Shell window by opening the Shell icon in the Workbench2.0 disk window.*



When you see a prompt, such as 1.SYS:>, you are ready to start entering commands. The Shell prompt may vary depending on your system. In this chapter, the prompt is represented by a 1>.

You must press Return at the end of each command line to execute the command. After the command is executed, the Shell prompt will reappear.

You can also execute AmigaDOS commands from the Workbench without opening a Shell window. The Execute Command item of the Workbench menu opens a requester that lets you enter a command. (This is described in Chapter 2, "Basic Operations.") You can enter and execute commands the same way as in a Shell. However, the Shell is more convenient when more than one command must be executed.

Getting Information About Disks

The next three steps illustrate the DIR, LIST and INFO commands, three common commands used to get information about the contents of your disks.

The DIR command generates a list of all the files and directories included on a disk or within a directory.

2. *Type DIR at the Shell prompt, and you should see the following output:*

```
1> DIR
Trashcan (dir)
Rexxc (dir)
Expansion (dir)
Libs (dir)
Monitors (dir)
WBStartup (dir)
Prefs (dir)
Fonts (dir)
C (dir)
Devs (dir)
S (dir)
L (dir)
Utilities (dir)
System (dir)
.info
Expansion.info
Shell.info
System.info
Utilities.info
disk.info
Monitors.info
Prefs.info
Trashcan.info
WBStartup.info
```

When a name is followed by (dir), it is a directory. Notice that there is a directory for each drawer that appears in the Workbench2.0 disk window. If you choose the Show All Files menu item, drawer icons for all the other directories will appear in the disk window.

The files with the .info suffix contain the data used to create the icons for the corresponding drawer or file. For instance, the Utilities.info file corresponds to the Utilities drawer icon. The .info files also contain any information entered in the icon's Information window, such as Tool Types or Default Tools.

3. *You can see a list of files included in a directory by typing DIR followed by the directory name. For instance:*

```
1> DIR Utilities
      .info          Clock
      Clock.info     Display
      Display.info   Exchange
      Exchange.info  More
      More.info      Say
      Say.info
```

This generates a list of the files, and any subdirectories, stored in the Utilities directory on the Workbench2.0 disk. You can also specify a disk or drive name with DIR to look at the contents of a specific disk, such as DIR DF0: or DIR Workbench2.0:.

If you want more information about the files and directories on the disk, use the LIST command. You'll see the same information generated by the DIR command, but the output will also contain the sizes of files, the **protection bits** for both files and directories, and the date and time the files or directories were created. The possible protection bits for a file or directory are listed below:

s	the file is a script
p	the file is pure
a	the file has been archived
r	the file is readable
w	the file can be written to
e	the file can be executed
d	the file can be deleted

If a protection bit is **set**, the appropriate letter is shown. This means that the bit is "on" and being used by the file. If the d bit is set, the file is deletable. If the bit is not set, a dash (-) appears. This means the bit is **clear**, or "off." If the d bit is clear, the file is not deletable. For more information on protection bits, see the "Protect" section of Chapter 8.

4. To list the contents of the Workbench2.0 disk, type:

```
1> LIST
Trashcan.info      1144 ----rwd 20-Jun-90 17:22:48
Trashcan           Dir ----rwd 20-Jun-90 04:35:07
Rexxc              Dir ----rwd 20-Jun-90 04:35:18
WBStartup.info     824 ----rwd 20-Jun-90 17:22:47
Utilities.info      824 ----rwd Thursday 16:03:05
System.info        824 ----rwd 20-Jun-90 17:22:47
```

A shortened version of the typical output is shown above.

Just as with the DIR command, you can specify a directory or drive name after LIST, and the contents of that directory or drive will be listed.

5. Type:

```
1> LIST Utilities
Say.info           486 ----rwd 12-Jul-90 16:03:00
Say                7124 ----rwd 20-Jun-90 17:21:47
More.info          454 ----rwd 12-Jul-90 16:02:59
More               11060 ---p-rwd 20-Jun-90 17:21:46
Display.info       563 ----rwd 12-Jul-90 16:02:57
```

A shortened version of the output is shown.

You can also use LIST to get information about a specific file. For instance:

6. Type:

```
1> LIST System/Setmap
setmap             4112 ----rwd 20-Jun-90 17:21:41
```

To get more general information about your system, you can use the INFO command. INFO displays a list of all the disks that are currently available to the system. Some of this information is similar to what is displayed in a disk window title bar, such as how much space is used and left on the disk. However, the INFO output also shows if there are any errors on the disk and whether or not the disk is write-protected.

7. Type INFO at the Shell prompt. Typical output will look like this:

```
1> INFO
```

Unit	Size	Used	Free	Full	Errs	Status	Name
RAM:	17K	17	0	100%	0	Read/Write	Ram Disk
DF0:	879K	1723	35	98%	0	Read/Write	Workbench2.0

Volumes Available:

Ram Disk [Mounted]

Workbench2.0 [Mounted]

Creating A New Directory

The MAKEDIR command creates a new directory on the disk. This is similar to choosing the New Drawer menu item, however a directory created with MAKEDIR does not automatically have an icon associated with it. You have to create the icon separately (see step 9).

8. To create a directory called Testdir on your Workbench2.0 disk, type:

```
1> MAKEDIR Testdir
```

A new directory is created, but you will not see a drawer icon for Testdir unless you choose the Show All Files menu item.

To create an icon for Testdir, you can make a copy of the .info file for any existing drawer icon. For instance, you can copy the

icon for the Expansion drawer by copying the `expansion.info` file. To do this you use the `COPY` command. You must be sure to specify both the name of the file you are copying (the source file) and a name for the copy (the destination file).

9. *Type:*

```
1> COPY Expansion.info Testdir.info
```

This makes a copy of the `Expansion.info` file and calls it `Testdir.info`. This file will be associated with the `Testdir` directory, and there will now be a `Testdir` drawer icon in the `Workbench2.0` disk window.

When you copied the `Expansion.info` file, you also copied the position of the icon. Therefore, the `TestDir` and `Expansion` icons will be on top of each other. Use the `Clean Up` and `Snapshot` menu items to rearrange your window.

When creating icons for new files or directories, be sure to copy an icon of the same type (drawer, project, or tool). Otherwise, you could have problems when you try to open the file from the `Workbench`.

Changing the Current Directory

Very often there are times when you want to perform several operations within a directory, such as copying, renaming and deleting files. Instead of typing the full path, including the directory name, with each command, you can change your Shell's **current directory**.

The current directory is AmigaDOS's reference point for that Shell, similar to the `Workbench`'s selected drawer window. At any given time in any Shell, there is always one directory that is the current directory.

There are two ways to think about the current directory:

- The path up to and including the current directory is assumed and does not need to be included in the path to a particular file.
- The current directory is the *default* directory—the directory AmigaDOS will work within if no directory is specified.

The current directory is the one in which the Shell will look for information first, before checking other places on the disk. When you open a Shell, the current directory is usually the root directory of your boot disk, known as SYS:.

To make another directory the current directory, use the CD command.

10. To make Testdir the current directory, type:

```
1> CD Testdir
```

Now, if you enter the DIR or LIST commands, the output will show the contents of the Testdir directory. When you want to work with files in Testdir, you simply have to type the file name instead of the complete path.

The current directory is part of the standard Shell prompt, such as:

```
1.Workbench2.0:Testdir>
```

This way you can always see what the current directory is before you issue any commands.

Regardless of the current directory setting, you can still refer to a file in any other directory by giving the full path. The current directory will not change unless you enter the CD command. You can change the current directory at any time according to what seems most convenient for the task at hand. Each Shell has its own independent current directory.

The Shell also supports the concept of an implied CD. You can CD to a directory by typing the name of the directory at the prompt. The Shell will look through the **search path** for a command of that name. (The search path is an ordered set of paths that AmigaDOS examines in order to find a command.) If it does not find a command of that name to execute, it will automatically CD to that directory.

When typing the directory name, be sure to specify it relative to the existing current directory. For instance, if the current directory is the root directory, Workbench2.0:, and you type:

```
1> Utilities
```

The Shell will search for a command called Utilities. When it does not find one, it will change the current directory to the Utilities directory. However, if you try to change back to the root directory by typing:

```
1> Workbench2.0
```

AmigaDOS will search forward in the directory structure for a directory called Workbench2.0. When it does not find one, it will respond with an Unknown command message. To CD to a higher directory, type the appropriate number of slashes or the full path. For instance, to CD back to the root directory, type:

```
1> Workbench2.0:
```

or, simply

```
1> :
```

If the current directory is Work:Program/Documents/Letters, typing:

```
1> //
```

will move you back to the Work:Program directory. You could also type:

```
1> Work:Program
```

Search paths are more fully explained in the next section.

Changing the Search Path

Just as changing the current directory can save you from having to type the full path to a command, so can adding directories to the search path. The current directory is always at the beginning of the search path, and the C: directory is always at the end.

Several other directories are usually added to the search path in the Startup-sequence, the file that is executed each time the Amiga is booted. The standard Startup-sequence adds the System, Utilities, Prefs, and S directories, as well as the Ram Disk, to the default search path. AmigaDOS will automatically look in these directories, in sequence, to locate a program to execute. If the file is not in any of those directories, the Object Not Found message will be displayed.

You can add any directory you like to the search path with the PATH command. Adding a directory to the search path eliminates the need to supply a path to the program. You simply need to enter the filename, and AmigaDOS will find it as if you had typed the complete path.

For instance, if you frequently use a program called Spell, in the directory Words, on a disk called English, you could add the English:Words directory to your search path. Normally, when you wanted to use the Spell file, you would have to specify the complete path: English:Words/Spell. If you have an extra disk drive added to your system, you could keep the English disk in that drive and add the Words directory to your search path. To do so you would type:

```
1> PATH English:Words ADD
```

Now when you wanted to access the Spell program, you could simply type `Spell` on the command line.

If you enter the PATH command in a Shell window, the new search path only applies to that Shell and any Shells opened from that Shell. Any other open Shell windows will still use the default path (unless you entered different PATH commands in those windows). To permanently add a directory to the search path, you must modify the User-startup or Shell-startup files. This is fully explained later in this chapter.

If you have two or more commands with the same name in different directories in your search path, AmigaDOS will always execute the first one found. This is based on the order of the search path. You can access the other command of the same name by specifying the full path to the command.



Working with Files

So far, your Testdir directory is empty. To get a file to work with, you're going to copy the Startup-sequence file in the S directory into your Testdir directory.

11. To copy the Startup-sequence file, type:

```
1> COPY S:Startup-sequence to Testdir/Textfile
```

This command makes a copy of the Startup-sequence file, in the S directory, in a file called Textfile in the Testdir directory. The original Startup-sequence stays in the S directory.

12. To make sure that the COPY command worked, type:

```
1> DIR Testdir
```

The output should show that Textfile is in the directory.

Some common commands you can use when working with files are COPY, TYPE, RENAME and DELETE. You've already used the COPY command when you copied the Startup-sequence file into the Testdir directory. The following steps show you how to use the other commands.

13. *The TYPE command lets you look at the contents of a file. Type:*

1> TYPE Textfile

and the contents of Textfile will appear on the screen. The text may scroll rather quickly. If you press the Space Bar the text will pause. To start the text scrolling again, press Backspace.

14. *To change the name of a file, use the RENAME command. You must specify both the old name and the new name.*

1> RENAME Textfile Document

This command will change the name of the file called Textfile to Document. The contents of the file will not be affected.

15. *To delete a file from the disk, use the DELETE command. Type:*

1> DELETE Document

The Document file will be deleted. The Testdir directory still exists even though it is empty.

You cannot delete the Testdir directory while you are working in that directory. You have to change directories (CD) to a higher-level directory, in this case the original Workbench2.0 root directory.

16. *A slash tells the CD command to move up one level in the directory hierarchy. Since you are working in the Testdir directory, type:*

```
1> CD /
```

The current directory will become the Workbench2.0 root directory.

If you were in a directory such as NewDisk:Testdir/Files, typing CD / would return you to the NewDisk:Testdir directory.

17. *To delete the Testdir directory, type:*

```
1> DELETE Testdir
```

The directory will be deleted, but the Testdir.info file will still remain in the Workbench2.0 disk's root directory.

18. *To delete the Testdir.info file, type:*

```
1> DELETE Testdir.info
```

Your Workbench2.0 disk will now have the same contents as when you started.

Working with Disks

There are several AmigaDOS commands that pertain solely to disks. When working through the Workbench, you can use the Format Disk menu item when you want to format a floppy disk or hard disk partition. The AmigaDOS command to allow you to do this through a Shell is called FORMAT. With FORMAT you must specify the location of the blank disk and a new name for the formatted disk.

19. Leave the Workbench2.0 disk in the disk drive, and type:

```
1> FORMAT DRIVE DF0: NAME EmptyDisk
Insert disk to be formatted in drive DF0:
and press RETURN
```

Remove the Workbench2.0 disk, and insert the disk to be formatted into the drive.



Do not press Return until you remove the Workbench2.0 disk and insert a blank disk. Otherwise, you could accidentally format your Workbench2.0 disk.

You must enter the DRIVE and NAME keywords with the FORMAT command. The DRIVE keyword specifies the disk drive that the disk is in, and the NAME keyword specifies a new name for the formatted disk.

To change the name of a disk, use the RELABEL command. RENAME only works with directories or files.

20. To change the name of the newly formatted disk called EmptyDisk to NewDisk, type:

```
1> RELABEL EmptyDisk: NewDisk
```

NOTE: If you have a floppy disk system with only one floppy drive, you should specify the disk by its volume name, not DF0: Using the drive name tells the Amiga to rename whatever disk is in that drive, and you could accidentally rename your Workbench disk.

If you have two disk drives or a hard drive, you could use the drive name instead of the volume name. For instance,

```
1> RELABEL DF2: NewDisk
```

To copy a disk through the Shell, use the DISKCOPY command. If you have one floppy drive, you will be prompted to swap disks just as if you had chosen the Copy menu item. However, the prompts will appear as messages in the Shell

window, not as system requesters with Cancel and Continue gadgets. The next step shows you how to make a copy of your Workbench disk.

21. Type:

```
1> DISKCOPY DF0: to DF0:  
Place SOURCE disk (FROM disk) in drive DF0:  
Press <RETURN> to continue
```

With the DISKCOPY command, you must type the word "to" between the name of the disk drives. If you have two disk drives, you could type:

```
1> DISKCOPY DF0: to DF2:
```

You can also use volume names, such as:

```
1> DISKCOPY Workbench2.0: to NewDisk:
```

Setting the Clock

Instead of using the Time editor to set the date and time, you can also use the AmigaDOS DATE command.

22. To see the currently saved date and time, type:

```
1> DATE  
Tuesday 17-Apr-90 11:34:58
```

If the output is incorrect, you can change it by specifying the correct date and/or time after the command. The acceptable format is DD-MMM-YY (date-month-year) for the date, and HH:MM:SS (hour:minute:second) for the time.

23. For instance, to set the date and time to July 22, 1992, 12:34 AM, type:

```
1> DATE 22-JUL-92 12:34:00
```


The DATE command alone does not save the time to the battery backed-up clock. If you were to reboot or power off, the date and time would be lost. To save the time to the system clock, you must use the SETCLOCK command.

24. Type:

```
1> SETCLOCK SAVE
```

Opening/Closing Shell Windows

Finally, two other commands that you should know are NEWSHELL and ENDSHELL. These commands let you open and close Shell windows.

25. Type:

```
1> NEWSHELL
```

A second Shell window will appear on the screen.

26. To exit that window, make sure it is active, and type

```
1> ENDSHELL.
```

The window will close.

Two other ways to close the Shell are by selecting the close gadget in the upper left corner or typing Ctrl-\. Use one of these methods to close your first Shell.

This section only touched upon the basic commands you can use while in the Shell. Be sure to read Chapter 8 to learn the full capabilities of the commands discussed in this section. Many of them have more advanced options that were not covered here.

Special AmigaDOS Characters

AmigaDOS has reserved several characters for special functions, such as adding comments to command lines, pattern matching, and redirecting the output of a command. These special characters are explained in this section.

Command Line Characters

The following characters can be used on the command line or in scripts.

Colon :

The colon is reserved for use after any device name (DF0:), volume name (Workbench2.0:), or assigned directory (SYS:). Do not put a space before the colon, or between the colon and any subsequent file or directory names.

If you did not type the colon, the system would look for a file or directory of that name. If you included a colon after a file or directory name, a requester would appear asking you to insert a volume of that name into any drive.

Slash /

A slash is used to separate directory and filenames in a path. For example:

```
1> LIST Reports/Salesreps/Eastern
```

The slashes separate the three directory levels, and specify that the subdirectory to be listed is Eastern.

A slash is also used to move up one level in the current directory structure. For example:

```
1> CD /
```

Using two slashes moves up two levels, and so on.

Semicolon

A semicolon is used to add comments to command lines. Anything to the right of a semicolon is ignored by AmigaDOS. This allows you to place descriptive comments on the same line with AmigaDOS commands. This is commonly used in documenting and debugging script files.

For example:

```
ASSIGN T: RAM:t ;set up T: directory for scripts
```

If the comment is too long to fit on one line, it can be continued on a second line. However, the new line must begin with a semicolon and is usually indented to the same level as the previous comment for readability.

Asterisk

Redirection is explained on page 7-33.

An asterisk is a convenient way to refer to the current window. It can be used as a FROM or TO argument or as a redirection filename — the source of input or the output destination. Pressing Ctrl-\\ will restore input/output to the default source.

For example:

```
1> COPY * TO Screenfile
```

copies whatever is typed into the current window to the file called Screenfile. To stop the copy, press Ctrl-\\.

Ctrl-\\ is also used to close a Shell window. Be careful not to press this key combination twice.

Back Apostrophe

A back apostrophe can be used to execute commands from within a string. When a string containing a command enclosed in back quotes is printed, the command will be executed. For example:

```
1> ECHO "DF0: contains `DIR DF0: ALL`."
```

prints DF0: contains followed by a directory of the disk in DF0:

Commands that refer to the current directory will not work correctly when invoked this way. The use of the back apostrophe automatically sets up a temporary sub-shell just for that command. Any reference to a current directory will be to the sub-shell's directory.

Pattern Matching

Pattern matching allows you to specify filenames by using special **wildcard** characters to match characters in the filenames. This lets you work on multiple files with one command. For instance, you can copy or rename all the files that begin with a specific letter, end with the same prefix, or reside in the same directory with one command.

Several different wildcards allow you to specify the type of match. In order to remove the special effect of these characters and search for a wildcard character, preface the character with an apostrophe ('). For instance, '?' will match ?, and " will match '. In the list below, a <p> indicates that either a single or multiple character string immediately adjacent to the wildcard will be matched. The wildcards are listed below:

?	Matches any single character.
#<p>	Matches zero or more occurrences of <p>.
<p1> <p2>	Matches if either <p1> or <p2> matches.
~<p>	Matches everything but <p>.
(<p1><p2>...)	Groups items together.
%	Matches the null string.
[<p>-<p>]	Delimits a character range.

For example:

A?B	Matches any three letter name beginning with A and ending with B, such as AcB, AzB, and alb.
A#BC	Matches any name beginning with A, ending with C, and having any number of Bs in between, such as AC, ABC, ABBC, ABBBC, and so on.
A#(BC)	Matches any name beginning with A and followed by any number of BC combinations, such as ABC, ABCBC, and ABCBCBC.
A(B C)D	Matches ABD or ACD
ABC#?	Matches any name beginning with ABC, regardless of what follows, such as ABCD, ABCDEF.info, or ABCXYZ.
#?XYZ	Matches any name ending in XYZ, regardless of what precedes it, such as ABCXYZ and ABCDEFXYZ.
[A-D]#?	Matches any name beginning with A, B, C or D.
^(XYZ)	Matches anything but XYZ.
^(#?XYZ)	Matches anything not ending in XYZ.
A(B D %)#C	Matches ABC, ADC, AC (% is the null string), ABCC, ADCC, ACCC, and so on.

*#? is equivalent to the * wildcard used by some other computer systems.*

The most frequently used combination is #? (matches any characters). This makes it simple to work with an entire group of related files, such as .info files. For example, to delete all the .info files in the Picture directory, you would type:

```
1> DELETE Picture/#?.info
```

Be careful when using #?. You could accidentally delete the contents of a disk.

Redirection

You can use the angle bracket characters (<) and (>) to **redirect** command input and output to a different destination.

Typically, the keyboard is used for command input, and the current Shell window is used for output. The redirection characters allow you to change the input/output to a specific file or device (printer, modem, etc.).

The redirection argument consists of the < (change input) or > (change output) symbol followed by a filename or device name. For instance, console output usually goes to the current Shell window. If you typed:

```
1> DIR >Testfile DF0:
```

the output will be sent to a file called Testfile. The output will not be shown on the screen. Notice that the redirection argument must come before any of the command's arguments. The angle bracket must be preceded by a space, but need not be followed by a space.

Similarly, you can change the source of the input from the keyboard to a file with the < symbol. For example:

```
1> DATE <Datefile ?
```

uses the contents of the Datefile file as the arguments for the DATE command. The command responds as if the contents of Datefile were typed at the keyboard.

It is only the console output of a command that is redirected, not the data the command works on. For example:

```
1> COPY >Log Picdir to PicsArchive: ALL
```

still copies all the files in the Picdir directory to the PicsArchive disk. However, the list of files being copied is sent to the Log file. Notice that the redirection argument appears immediately after the command and before the other arguments.

You can also redirect output and append material to an existing file. To do this, you must use two output symbols together (>>) with no spaces between them. For example:

```
1> Postscript >>Laser/Letter
```

executes the program Postscript, adding its output to the end of the Laser/Letter file.

The proper use of the redirection arguments depends on the syntax of the command involved. In some cases, > or < could replace the keywords TO or FROM.

Redirection only applies to the command line in which the redirection characters are used. AmigaDOS reverts to the default input and output sources for any subsequent commands.

Features of the Shell

As shown in the previous section, the Shell allows you to communicate directly with AmigaDOS. It uses a special type of window called a **console window**. A console window is a text-only interface, which means that it accepts typed input from the keyboard. You cannot use icons in a console window. Special features of the console window are described below:

- All of the standard Workbench window gadgets can be used on the Shell window except for the scroll gadgets.
- When you select the Shell window's zoom gadget, the window expands to fill the entire screen.
- The System Default Text font, as specified by the Font editor, is used in the Shell window. This must be a nonproportional font, such as Topaz or Courier.
- Workbench background patterns do not appear.
- The WBConfig option of double-clicking to bring a window to the front does not apply.

You can have several Shell windows open at once. Each window is independent. While commands entered in one Shell are being executed, you can enter and execute different commands in another Shell window.

The Shell environment offers command-line editing and **command history**. These features allow you to use the cursor keys to fix typing mistakes or repeat commands typed earlier.

Editing

There are several keys and key combinations you can use to edit the current command line. They include standard text-editing keys, plus several Control-key sequences summarized below:

left cursor	Moves cursor one character to the left.
right cursor	Moves cursor one character to the right.
Shift-left cursor	Moves cursor to the beginning of the line.
Shift-right cursor	Moves cursor to the end of the line.
Backspace	Deletes the character to the left of the cursor.
Del	Deletes the character highlighted by the cursor.
Ctrl-H	Deletes the last character (same as Backspace).
Ctrl-M	Processes the command line (same as Return).
Ctrl-J	Adds a line feed.

Ctrl-W	Deletes the word to the left of the cursor.
Ctrl-X	Deletes the current line.
Ctrl-K	Deletes everything from the cursor forward to the end of the line.
Ctrl-Y	Replaces the characters deleted with Ctrl-K.
Ctrl-U	Deletes everything from the cursor backward to the start of the line.

In addition to command line editing, the Shell also provides command history, which allows you to recall previously-entered command lines, edit them, and re-enter them. This is useful when you want to repeat a command or enter several very similar commands.

The Shell uses a 2K command-line buffer to retain command lines. The exact number of lines varies depending on lengths of the lines actually stored. When the buffer fills up, the oldest lines are lost. You access lines in the buffer through the up and down cursor keys:

up cursor	Moves backward in the history buffer (earlier lines).
down cursor	Moves forward in the history buffer (later lines).

For example, if you wanted to copy several .info files from one directory to another, you could enter the full line with the complete path once, then recall the line as many times as necessary, changing only the filename portions of the line.

You can also search for the most recent occurrence of a specific command by typing the command line, or the beginning of it, then pressing Shift-up cursor (or Ctrl-R). For instance, if you type DIR and press Shift-up cursor, you will be returned to the last command to perform a DIR of any directory. Pressing Shift-down cursor moves you to the bottom of the history buffer, leaving the cursor on a blank line.

Some additional keystrokes you can use in the Shell are:

Space bar (or any printable character)	Suspends output (stops scrolling).
Backspace	Resumes output (continues scrolling).
Ctrl-C	Sends a BREAK command to the current process (halts the process).
Ctrl-D	Sends a BREAK command to the current script (halts the script).
Ctrl-S	Suspends output.
Ctrl-Q	Resumes output if it was suspended with Ctrl-S.
Ctrl-\	Closes the Shell window.

Another feature of the Shell is the ability to **type ahead**. If you begin typing while output is occurring in a Shell window, the output will pause. If you press Return, the output will resume, and the recently typed line will be used as the next line of input. To resume the output without executing the typed line, delete your input. The text will resume scrolling as soon as the last character is erased.

Copying and Pasting

A new feature of the Shell is the ability to **copy and paste** information from one console window, such as a Shell or ED window, to the same or another window. For instance, if you are using a text editor to write a script file, you can generate a DIR listing in a Shell, then transfer it to the editor. This saves having to retype all the information.

To copy and paste information, you use the mouse to highlight the area of text to be copied. This is the only Workbench-style mouse operation performed in Shell windows. To highlight the text, move the pointer to the beginning of the text area, hold down the selection button, and drag the pointer to the end of the area to be copied. The text will be highlighted in the window as you drag the mouse. Release the selection button, and the area you have indicated will remain highlighted. Press right Amiga-C to copy the highlighted text into memory.

Now move the pointer to the other console window, and click inside it to activate it. Move the cursor to the point where you want to insert the text. Press right Amiga-V, and the text will be copied to the second window. You can paste the text repeatedly by moving the cursor to the desired location and pressing right Amiga-V.

Customizing the Window

The WINDOW Tool Type is also supported by ED and ICONX.

The Shell supports a WINDOW Tool Type that allows you to specify the size, position, and features of the Shell window. The Tool Type is in the form of:

WINDOW = CON:x/y/width/height/title/option

where:

x	Is the number of pixels from the left edge of the screen to the left border of the window.
y	Is the number of pixels from the top of the screen to the top of the window.
width	Is the width of the window, in pixels.
height	Is the height of the window, in pixels.
title	Is the text that appears in the window title bar.

The permissible options are:

AUTO	The window automatically appears when the program needs input or produces output. With the Shell window, it will open for input immediately. The window can only be closed with the ENDSHELL command. Selecting the Shell's close gadget will close the window, but it will re-open immediately since it is expecting input.
CLOSE	The window has all the standard gadgets, including a close gadget.
BACKDROP	The window appears on the backdrop, behind all the Workbench windows. The only gadget in the window border is the zoom gadget. This Shell window cannot be brought to the front of the screen; you have to resize the Workbench windows to see it.
NOBORDER	The window opens without any left or bottom window border. Only the zoom, depth, and sizing gadgets are available.
NODRAG	The window cannot be dragged. It has a zoom, depth and sizing gadget, but no close gadget.
NOSIZE	The window only has a depth gadget.
SCREEN	The window will open on a public screen. The screen must already exist. You must specify the name of the screen after the SCREEN keyword.

SIMPLE	If you enlarge the window, the text will expand to fill the newly available space, allowing you to see text that had been scrolled out of the window.
SMART	If you enlarge the window, the text does not expand to fill the newly available space.
WAIT	The window can only be closed by selecting the close gadget. (An example of this is the Execute Command Workbench Output Window.)
WINDOW	<i>NOTE: THIS IS FOR PROGRAMMERS ONLY.</i> You can specify the hex address of an already open Intuition window for the console handler to use instead of opening a new window. The correct format is WINDOW0xhhhhhh.

For instance, if you were working with a Hires screen, and you wanted your Shell window to fill the entire screen, have a close gadget, include your name in the title, you would use the following Tool Type:

WINDOW = CON:0/0/640/200/Ralph'sShell/CLOSE

Remember, if you add the x and width numbers, they cannot be greater than the width of the screen. Likewise, the sum of the y and height numbers cannot be greater than the height of the screen.

Closing the Shell

When you have finished your work in a Shell window, close it. Any open window, even a small one, takes up a certain amount of memory, including Chip RAM.

Before you can close a Shell window, any programs that were run from the Shell must have finished. If a program is still running, the text cursor will have moved down a line, but there will be no prompt string to its left. You can still type into the window, but there will be no response.

There are three ways to close a Shell window:

- select the close gadget
- type the ENDSHELL command
- press Ctrl-\

The Shell-startup File

Whenever you open a new Shell, the S:Shell-startup file is executed. The Shell-startup file allows you to customize your Shell environment. You can edit Shell-startup to set up command aliases and to change the Shell prompt.

Using Aliases

An **alias** is an abbreviation for a long and/or frequently used command. An alias can be either local or global. Local aliases are entered in a Shell window and are only recognized in that Shell. Global aliases are entered into the Shell-startup file and are recognized by all Shells.

The angle brackets, < >, indicate that information must be substituted. For instance, <name> means that you must enter a name after the alias command. Do not type the brackets.

An alias takes the form of:

```
ALIAS <name> <string>
```

where <name> is the alias, the name you want to type at the Shell prompt to execute a command. The <string> is the command line you want to execute. Some example ALIAS commands are shown below:

```
ALIAS d0 DIR DF0:
ALIAS d1 DIR DF1:
ALIAS edus ED S:User-startup
ALIAS edsh ED S:Shell-startup
ALIAS del DELETE
ALIAS cp COPY
ALIAS ren RENAME
```

Whenever you use the <name> at a Shell prompt, the <string> will be substituted, as if you had entered it instead. For instance:

```
ALIAS d0 DIR DF0:
```

lets you type `d0` to display the contents of the disk in DF0:

The <string> can include arguments as well as a command, but it must *begin* with an AmigaDOS command. The alias must be entered immediately after the prompt, but you can include further arguments on the line after the alias.

For instance, if you are using the alias shown above,

```
ALIAS d0 DIR DF0:
```

typing:

```
d0 System
```

would generate a directory of the System directory on the disk in DF0:.

See the "ALIAS" section of Chapter 8 for more information.

Changing The Prompt

The PROMPT command lets you customize the Shell prompt. By default, it shows the process number, a period, the current directory, a right angle bracket (>), plus a space:

```
1.SYS:>
```

This is represented in the Shell-startup file by:

```
""%N.%S> "
```

where %N represents the current Shell number and %S represents the current directory. The entire string is enclosed in quotes to maintain the final space after the >.

You can have the prompt display almost anything you want, with or without the process number and directory information. The prompt can contain **escape sequences**, which allow you to change text color and style in the prompt string or clear the screen. (Escape sequences are explained in the following section.) You can even embed complete AmigaDOS commands in the prompt by using the back apostrophe character (').

See the "PROMPT" section of Chapter 8 for more information.


Using Escape Sequences

Escape sequences can control how the text appears in a console window, such as the text color, style (bold, italics, underline) and margins. AmigaDOS recognizes standard ANSI X3.64 sequences when they are typed on the command line or embedded in strings. Escape sequences consist of one or more characters, sometimes with a numerical argument, prefaced by the escape character. Spaces are not normally used in the sequence of characters.

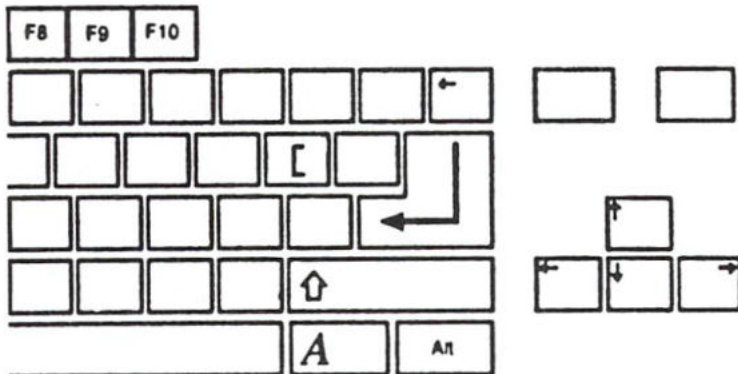
The table on page 7-45 shows the most common escape sequences used in a Shell. The escape sequence is shown using the following format:

Esc[#X

where:

- | | |
|-----|--|
| Esc | Represents the Escape key. Simply press Esc. Do not type the letters e, s, and c. When you press Esc, a reversed open bracket  appears in the console window. |
| [| Represents the open bracket key. |
| # | Represents a numerical argument. |
| X | Represents an alphabetic key. Escape codes are case-sensitive. If an uppercase letter is shown, press Shift and the key. If a lowercase letter is shown, just press the appropriate key. |

If your country's keyboard does not have an open bracket key, you must press Alt plus the key shown below, regardless of what is shown on the keycap:



Standard Escape Sequences for Console Windows

Sequence	Action
Esc	Clears the window and resets all modes to defaults
Esc[0m	Resets graphics modes to defaults
Esc[1m	Makes text boldface
Esc[3m	Makes text italic
Esc[4m	Underlines text
Esc[7m	Makes the text reverse video
Esc[8m	Makes text match background color
Esc[22m	Turns off boldface
Esc[23m	Turns off italics
Esc[24m	Turns off underlining
Esc[27m	Turns off reverse video
Esc[28m	Returns the text color to normal
Esc[30m	Makes text color0 (background, default grey)
Esc[31m	Makes text color1 (shadow, default black)
Esc[32m	Makes text color2 (shine, default white)
Esc[33m	Makes text color3 (accent, default blue)
Esc[3#m	Makes text color# (4-8)
Esc[39m	Makes text default color (color1)
Esc[40m	Makes text background color0 (default grey)
Esc[41m	Makes text background color1 (default black)
Esc[42m	Makes text background color2 (default white)
Esc[43m	Makes text background color3 (default blue)
Esc[4#m	Makes text background color# (4-8)
Esc[49m	Makes text background the default color (color1)
Esc[#u	Sets maximum length of lines in window to #
Esc[#t	Sets maximum number of lines in window to #
Esc[#x	Starts text # pixels from left window border
Esc[#y	Starts text # pixels from top of window

The escape sequence is executed when you press Return or when the string containing the sequence is printed. When entering escape sequences in a string, you can use the character combination `*E` to represent the pressing of Esc. For example, you can add the following line to your Shell-startup file to give all Shell windows a boldface, color3 prompt string:

```
PROMPT ""*E[1m*E[33m%n.%s> *e[0m"
```

Notice the use of `*E` to represent Esc. A space is entered after the `%s>` to allow a space between the prompt and your input. The final sequence, `*E[0m`, turns off the previous modes so that only the prompt is affected by the boldface and color3 codes. The Shell-startup file contains several examples of escape sequences.

Running Programs

Most programs can be run both from the Workbench and from the Shell. To run a program from the Shell, you usually type the program name at the Shell prompt. (If the program file is not in the current directory or search path, you will have to specify the complete path to the file.) This tells AmigaDOS to load and execute the program.

When using application software, be sure to read the license agreement enclosed with the software to determine if you can make a backup copy of the program disk(s). If it is allowed, use the backup(s) as your working disk(s) and store the originals in a safe place. Store your data files on different disks. Leaving the program disk write-protected helps prevent disk corruption by errors or software viruses. If you must store data and programs on the same floppy, it is vital that you keep backup copies of the unmodified, original program disks.



Most programs allow you to specify additional information on the command line after the program name, such as the name of a file to load or startup options. This is called **argument passing** (giving a command parameters to follow).

For example:

```
1> MEmacs
```

loads and runs MEmacs.

```
1> MEmacs S:User-startup
```

loads and runs MEmacs, automatically opening the User-startup file in the S: directory as the file to begin editing.

```
1> SYS:Utilities/Clock WIDTH 200 HEIGHT 100 SECONDS
```

loads the Clock with a specified size of 200 pixels by 100 pixels and the Seconds option turned on.

Often this argument-passing ability is provided as a convenience, allowing you to specify right on the command line what might otherwise be accomplished from within the program with a menu operation or two. However many programs, especially those which can only be run from a Shell, require that filenames or other arguments be specified on the command line with the program name.

Another way to enter a program name is with the RUN command which loads and runs a program in the background without opening a Shell window. This means that the Shell prompt will return after the program is opened.

For instance, if you type:

```
1> MEmacs
```

the MEmacs editor will open, but you will not be able to enter any additional commands or close the Shell window until you exit MEmacs.

But, if you type:

```
1> RUN MEmacs
```

the MEmacs editor will open, and the Shell prompt will return. You can now enter additional commands.

When a program is invoked with RUN, a message indicating the new process number will be displayed, such as [CLI 2]. Any output that the program generates will appear in the originating Shell window.

You cannot close the Shell window if any programs launched from that window are still running. For instance, if you had opened MEmacs through the Shell, you could not close the Shell window until you had exited MEmacs. One way to avoid this situation is by typing:

```
1> RUN >NIL: MEmacs
```

You'll learn more about the NIL: device in "The Startup-Sequence" section on page 7-55.

Scripts

A script file, also called a command file, is a text file containing a list of commands, typed on separate lines much as they would be typed at successive Shell prompts. A script can be created with any text editor that saves files in ASCII format, such as ED or MEMacs.

Scripts are used for repetitive and/or complex tasks. For instance, it is sometimes necessary to perform the same operation on a large number of files. Instead of entering each command individually, you could create a script that repeats the same command, but substitutes a different filename in each command line.

For instance, if you had several files that you needed to rename, you could use a script like this:

```
RENAME section1 chap1.1
RENAME section2 chap1.2
RENAME section3 chap1.3
RENAME section4 chap1.4
RENAME section5 chap1.5
RENAME section6 chap1.6
RENAME section7 chap1.7
RENAME section8 chap1.8
```

This example assumes that the files are in the Shell's current directory. If not, you would have to specify the complete path to the file.

Once a script file has been created, it is run via the EXECUTE command. Typing EXECUTE <script> at a Shell prompt, tells the system to read the script and execute each line.

If the s (script) protection bit is set, you can enter the script name without preceding it with EXECUTE.

Scripts can even halt and request information from the user before continuing, so that variable conditions can be accommodated with a single script. For instance, the following script could be used to copy files from a hard drive to a floppy disk. In this instance, only 6 files fit on a floppy disk. After 6 files are copied, the user is asked whether to continue the copy. This also allows time to put a new disk into the disk drive.

```
COPY 2k.eps DF0:
COPY 2m.eps DF0:
COPY 2n.eps DF0:
COPY 2o.eps DF0:
COPY 2t.eps DF0:
COPY 2v.eps DF0:
ASK "Continue Copy?"
IF WARN
  COPY 3a.eps DF0:
  COPY 3c.eps DF0:
  COPY 3g.eps DF0:
  COPY 3aa.eps DF0:
  COPY 3bb.eps DF0:
  COPY 3ff.eps DF0:
ENDIF
```

If Y is pressed at the Continue Copy? prompt, the script will go on to copy the remaining files to a disk in DF0:. If N is pressed, the script will be terminated.

Several AmigaDOS commands are specifically for use in scripts. These are listed below:

ASK	Asks for user input.
ECHO	Prints a string.
ELSE	Allows an alternative in a conditional block.
ENDIF	Terminates an IF block.
ENDSKIP	Terminates a SKIP block.
FAILAT	Sets the failure condition of the script.
IF	Handles conditional operations.

LAB	Specifies a label; used in conjunction with SKIP.
QUIT	Specifies a return code that will cause a command to exit.
SKIP	Execution of the script skips ahead to the specified label.

By including special characters known as dot characters in your script, you can specify places for parameter substitution. These parameters can be entered as arguments to the EXECUTE command. Please read the appropriate sections of Chapter 8 for more information on these commands.

Condition Flags

Every command sets a certain **condition flag**, a condition upon which the command will fail. When a command is executed, a **return code** indicates if a command was successful or if it failed. The standard return codes are:

- 0 The command was successful.
- 5 Represents a caution and indicates that some type of error occurred, even though it was not serious enough to abort the command. If the command is part of a script, subsequent commands will be carried out. Several commands set the condition flag to WARN to specify the outcome of a command. For example:

```
1> INSTALL DF0: CHECK
```

checks the disk in DF0: to see if it is bootable or not. If it is, the condition flag is set to 0; if not, it is set to 5.

- 10 Represents an error. In scripts a return code of 10 will abort the script, unless a higher limit has been set with the FAILAT command.
- 20 Represents a failure.

Several commands, such as INSTALL and SEARCH specifically use the WARN flag to signal certain conditions for testing in scripts.

For instance, in the example Copy script on page 7-50 the ASK command prompts the user as to whether they want to continue the copy:

```
ASK "Continue Copy?"  
IF WARN  
COPY 3a.eps DF0:
```

If Y is pressed, the condition flag is set to 5 (WARN), and the IF block will be carried out. If N or Return is pressed, the condition flag is set to 0 (NO ERROR), and the script will be aborted, since the IF statement did not receive the specified return code.

Other values may be returned by different software. In these cases, the values listed above are considered lower limits of the specified condition. You can interpret the values as:

0-4	No Error
5-9	Warn
10-19	Error
20 or above	Failure

Environment Variables

Environment variables are variables that are maintained by AmigaDOS itself, rather than individual applications, which means that the variables can be accessed and used by different

programs or scripts. For instance, AmigaDOS maintains Workbench and Kickstart variables that track the current version numbers of your Workbench and Kickstart software. The following line, executed in the Startup-sequence script, causes the Workbench version number to be printed on the opening screen:

```
ECHO "Amiga Workbench Disk. 2.0 Release Version $Workbench"
```

When a variable preceded by a dollar sign(\$) is encountered in a script, the variable name is replaced by the value assigned to the variable. Then the line is executed as if you had originally entered the value.

You can create environment variables with the SET and SETENV commands. SET lets you create **local** variables. A local variable is only recognized by the Shell in which it is created and any Shells created by that original Shell. For instance, if you are creating an environment variable in your Shell window, then execute the NEWSHELL command through the Execute Command menu item, the new Shell will not recognize any of the variables created in your original Shell. However, if you had opened a second Shell by typing the NEWSHELL command in your original Shell, the new Shell will recognize any variables created in its parent Shell. You can use the GET command to display the value associated with a variable, and the UNSET command to remove a variable.

SETENV creates **global** variables which are recognized by all Shells. Global variables are stored in the ENV: directory. You can use GETENV to display the value associated with a variable, and UNSETENV to remove a global variable. It is generally safest to use global variables only when you specifically need certain values to be available to other processes.

EVAL is an AmigaDOS command that evaluates simple expressions. It is explained in Chapter 8.

A common reason for using an environment variable in a script is to hold string information. The string might be long and tedious to type, making it easier to substitute a variable for it. Or the value for the string may change, making it easier to change the value of the variable than to continuously re-edit the script.

When given a numerical value, an environment variable can be used in calculations and expressions just as if it were the number it represents. For instance, you could assign the value 3.14159 to a variable called pi, then use \$pi in EVAL expressions. For instance:

```
1> EVAL 5.8 * $pi
15
```

A few applications support the use of environment variables. For instance, the More program in the Utilities directory supports an Editor environment variable. You can use SETENV to specify MEmacs as your editor of choice:

```
1> SETENV Editor Extras2.0:Tools/MEmacs
```

Be sure to specify the complete path to MEmacs. If you have a hard disk system, this will be Sys:Tools/MEmacs.

If you were using More to look at the contents of the Startup-sequence file, you could press Shift-E, and you would automatically be transferred to an MEmacs screen with the Startup-sequence loaded and ready for editing.

Some variables, like the Workbench and Kickstart variables explained above, have already been created. The Shell responds to the Echo variable and maintains the Process, RC, and Result2 local variables automatically. These are explained below:

ECHO	Controls whether commands are echoed to the screen before being executed.
PROCESS	The process number.

RC	The return code of the last command executed, 0, 5, 10, or 20. This is often used in scripts.
RESULT2	The secondary return code, or error number, that explains why a command failed. For instance, an error number of 205 indicates an Object not found error. You can use the FAULT command to interpret the error number. This is also used in scripts.

For instance, if you include the SET ECHO ON command at the beginning of a script, each line of the script will be echoed to the screen as it is executed.

The Startup-Sequence

Whenever you boot your Amiga, the Startup-sequence script is executed. This file is located in the S: directory. The Startup-sequence file can allocate disk buffers, make device assignments, set command aliases, and perform any other functions that can be accomplished with AmigaDOS commands.

The Startup-sequence, and the other startup files in the S: directory, can easily be modified to customize many aspects of your system. In addition to more technical system matters, the startup files can run programs at startup, print special introductory messages, or automatically open a Shell window on the Workbench screen.

There are several levels of importance to the items you will find in the Startup-sequence. Some are not necessary to the

It is recommended that you create a User-startup file in the S directory. You can then make additions and changes to the startup procedure in the User-startup file instead of to the Startup-sequence file.

functioning of the Amiga. Some are only needed if you will be using certain resources, and some must appear for the system to function properly.

An example of a typical Startup-sequence file is shown below. The line numbers have been added to help you reference the explanatory paragraphs that follow.

```

1 version >NIL:
2 failat 21
3 setlock >NIL: load
4 resident >NIL: c:ilist pure add
5 resident >NIL: c:copy pure add
6 resident >NIL: c:assign pure add
7 copy >NIL: ENVARC: ram:env all quiet noreq
8 nakedir ram: ram:clipboards
9 assign T: ram:T ;set up T: directory for scripts
10 if exists sys:monitors
11 list >T:non-start sys:monitors/(<?>.info) lformat="run >NIL: sxsx"
12 execute t:non-start
13 endif
14 assign ENV: ram:env
15 run >NIL: iprefs >NIL:
16 wait >NIL: 5
17 addbuffers >NIL: df0: 15
18 echo "Amiga Workbench Disk. 2.0 Release Version SWorkbench"
19 BindDrivers
20 setenv Workbench SWorkbench
21 setenv Kickstart SKickstart
22 resident c:execute pure add
23 assign LIBS: ram:clipboards
24 assign DEVS: s:
25 mount speak:
26 mount sfx:
27 mount pipe:
28 path ram: c: sys:utilities sys:rexxc sys:system s: sys:prefs sys:ubstartup add
29 if exists sys:tools
30 path sys:tools add
31 endif
32 rexxmnt >NIL:
33 if exists s:user-startup
34 execute s:user-startup
35 endif
36 LoadMB
37 endcill >NIL:

```

Before examining each command line, there are some basic elements of the Startup-sequence with which you should be familiar.

Many of the commands end in >NIL:. This is an example of the use of the redirection operator, >, to send the output of a command to the NIL: device. The NIL: device is a dummy destination used to eliminate the output of a command when it is not needed. Redirecting output to NIL: does not prevent the command from being executed or operating normally. It only inhibits the display of any text a command's execution may produce in the Shell window. It is commonly used in the Startup-sequence to prevent the appearance of messages and requesters on the opening screen.

A common use of the Startup-sequence is to **assign** directories. The ASSIGN command assigns a logical device name to a directory. Most of the basic system directories are assigned to device names, such as C:, LIBS:, DEVS:. This is necessary since

many software applications assume that they will be able to find the system files via the assigned device names.

For instance, if you use a desktop publishing program, it probably looks for the various fonts it supports in the FONTS: directory. The actual directory containing the fonts could be Workbench2.0:Fonts, System2.0:Fonts, even MyDisk:Fonts. However, by using ASSIGN to assign the directory to the logical name FONTS:, the desktop publishing program will be able to locate the necessary files.

Some applications expect you to make certain assignments in your Startup-sequence to allow the program to run and access specific files. This is especially common if you are installing the program on a hard disk. If this is the case, please see Chapter 6, "Using a Hard Disk," for more detailed instructions on how to do this.

The paragraphs below contain brief explanations of each of the lines in the standard Startup-sequence. For more information on the individual commands, see Chapter 8, "AmigaDOS Reference."

- 1 The VERSION command checks the internal version numbers of both Workbench and Kickstart. This ensures that a Workbench2.0 disk boots on a Version 2.0 system only.
- 2 FAILAT specifies the fail level for the script. In this case, the fail level is 21 which means that the script will continue even if an individual command cannot be executed.
- 3 The SETCLOCK command sets the date and time from the hardware clock. The Amiga has two clocks: the battery backed-up hardware clock and the software clock. The hardware clock runs all the time, even when the Amiga is turned off. However, it is the software clock that tells applications the current time.

The SETCLOCK LOAD command reads the time from the hardware clock and loads it into the software clock. It also displays the currently set date and time on the opening screen.

All Amigas have software clocks. Some A500s do not have battery-backed hardware clocks.

NOTE: If you have a system that does not have a built-in hardware clock, you may wish to insert an ECHO statement into your Startup-sequence to remind you to set the correct time with the Time editor. When a SETCLOCK command is run on a machine without a hardware clock, the message Battery Backed-up Clock not found is displayed.

Internal commands are already in ROM and do not have to be made resident.

- 4-6 The RESIDENT command loads an AmigaDOS command into memory so that it does not have to be loaded from disk again. In this case, it is making the LIST, COPY and ASSIGN commands **resident**. This makes execution of the command much faster as it does not have to be read from either floppy or hard disk before execution. Making a command resident is different from just copying a command to RAM:. If you simply copied the command to RAM:, disk routines would still have to be used to access the files, and the commands would still be copied to another place in memory before execution. Making commands resident is usually reserved for commands that are executed repeatedly.

NOTE: Only those commands with the p (pure) protection bit set should be made resident.

- 7 This line copies the Preferences settings saved in ENVARC: to the RAM:Env directory. RAM:Env is where programs will look for the current Preferences settings.
- 8 Directories in RAM: must be created every time the Amiga is booted. The MAKEDIR command creates the T and Clipboards directories. The T directory is used for temporary storage of scripts, including backup files created by many programs. The Clipboards directory is used to hold information that is being exchanged between programs.
- 9 Certain directories, such as the T directory, should be ASSIGNed. The command ASSIGN T: RAM:t assigns the RAM:t directory to the logical device T:. This makes it easy for programs that use the T directory to find it. Instead of looking for RAM:t or SYS:t, the programs just look for T:.

10-13 This is an example of the use of a conditional operation in a script. The IF statement, IF exists sys:monitors, tells the system to look for the SYS:Monitors directory. If it exists, the subsequent commands will be carried out. If not, the script will skip ahead to the line following the ENDIF command.

If SYS:Monitors exists, the LIST command is used to generate a short script which automatically runs the AddMonitor program for any monitor icons found in the Monitors drawer.

14 This is another example of an ASSIGN statement. The RAM:Env directory, which contains the Preferences settings, is being assigned to ENV:. This assignment is necessary for Preferences to work properly.

15 IPrefs is a program that communicates certain Preferences information, stored in the ENV:sys directory, to the system.

16 The WAIT command tells the system to wait the specified amount of time to allow the previous command to finish. The script will wait 5 seconds to give IPrefs time to tell the system what screen resolution you want before opening the first window.

17 Every floppy and hard drive must have a certain number of memory **buffers** allocated to it in order to function. The ADDBUFFERS command can be used to speed up disk related activity within the system. ADDBUFFERS allows you add additional buffers to a drive. (The default is 5.) In this instance, 15 buffers are added, resulting in a total of 20.

The more buffers allocated to a drive, the faster the response time when accessing data on the disks. However, if you add too many buffers, you may consume too much free memory.

For most floppy drives, 15-30 buffers is adequate. You can add more if you have extra memory. Using too few buffers will slow disk performance noticeably. ADDBUFFERS is most useful early in the startup sequence so that subsequent commands loaded from disk can be accessed faster.

A buffer is a temporary storage area in RAM.

- 18 The ECHO command is used to display a message on the screen. In this case, the opening screen message and Workbench version are displayed.

ECHO commands can be placed anywhere within a script. For instance, if you want the computer to remind you of something each time you boot, you can insert an ECHO statement into the User-startup file, and the message will be displayed on the screen. If you have several different boot floppies, you can insert an ECHO statement into each disk's User-startup file to remind you of which one you are using.

- 19 BINDDRIVERS is used to load and run device driver software for any expansion devices, such as hard disk controllers, attached to your system. The device drivers are supplied by the manufacturer of your device and must be copied into your Expansion drawer. If you do not have any expansion devices, BINDDRIVERS has no effect. However, if the BINDDRIVERS command is not given and you do have expansion devices, your system won't recognize the presence of any devices that do not auto-configure. The BINDDRIVERS command must appear before any other commands referencing the expansion devices.
- 20-21 The SETENV command creates the global environment variables specifying which versions of the Workbench and Kickstart software you are using.
- 22 This is another example of the RESIDENT command. In this case, the EXECUTE command is being made resident.
- 23-24 This is another example of the ASSIGN command. In this case, RAM:Clipboards is being assigned to the logical volume CLIPS: This is a common directory used by applications for exchanging data with other programs. The S: directory is being assigned to the logical volume REXXC: for programs that use AREXX.

- 25-27 The MOUNT command is used to inform the Amiga that an additional device has been added to the system. The device name, such as AUX: or SPEAK:, must be entered with the command, and an entry for that device must exist in the **MountList** file in the DEVS: directory. MOUNT commands must appear before any other commands that reference that device.
- 28 The PATH command specifies the default search path that the system will follow when looking for a command or filename. Any directory names that are in the search path do not have to be typed at the Shell prompt. The RAM: directory and all of the Workbench directories are added to the search path. If you wanted to run the Clock from the Shell, you could just type Clock instead of SYS:Utilities/Clock.
- 29-31 This IF-ENDIF statement checks to see if the Tools directory is in the SYS: directory, as it is for hard disk users. If it finds Tools, the directory is added to the search path.
- 32 Rexxmast is a background program used by AREXX. It must be active before any AREXX programs can be run.
- 33-35 This IF-ENDIF conditional checks for the existence of a script called User-startup in the S: directory. If such a script exists, it is executed. This is an easy way for you to add additional commands to your Startup-sequence, such as making more commands resident, assigning application programs to a hard disk drive, and adding frequently used directories to the search path.
- 36 LOADWB loads the Workbench into the Amiga's memory. Without this command, the Workbench screen and window would not appear. LOADWB should always be at the end of the Startup-sequence since it depends on the preceding commands having set up the proper search path, directory assignments, Preferences settings, etc.

- 37 Endcli >NIL: closes the initial Shell window, leaving the Workbench screen displayed.

Any AmigaDOS command, not just those mentioned above, can appear in a startup script. Be sure to read Chapter 8 for complete specifications about the individual commands so that you understand how to use them.

Editing the Startup-Sequence



In most cases, it is strongly suggested that you do not alter the original Startup-sequence file. Instead, create a new file called User-startup that contains any additional commands you want to add to the startup process.

If you do make changes to your Startup-sequence file, *make sure you are working on a copy of your Workbench disk, not the original*. If you make a mistake, the execution of the Startup-sequence is aborted, and you will be left with only a Shell prompt. Depending on whether the error occurred before or after LOADWB, you may not be able to access any menus or icons. Normally, the FAILAT 21 command ensures that the Startup-sequence will complete execution.



If you have a hard disk, be sure to copy your original, unaltered Startup-sequence file to a backup floppy. If you make a fatal error editing your Startup-sequence, you will be able to recover by copying the original file back onto your hard disk.

If an error occurs before the PATH command is executed, only the C: and SYS: directories will be in the search path. You will have to specify the complete path to any other directories you want to access. If this occurs, reboot with a good Workbench disk, and return to the edited Startup-sequence to try and discover the error. Be sure to keep the original Workbench disk write-protected at all times so you do not alter it by mistake.

There are some things you should keep in mind when editing your User-startup or Startup-sequence file:

- Be sure you understand the correct command syntax as shown in Chapter 8. Try out any commands you plan to insert into the Startup-sequence in a Shell window first. If a command works properly in the Shell, it will probably work as expected in the Startup-sequence.
- Pay attention to the order of commands in the script. Some commands, like ECHO and SETCLOCK, can be put anywhere. However, when inserting commands that refer to directories and files, be sure you aren't referencing something that has not yet been created, assigned, or given a valid path. For instance, if you insert a command copying something to the T: directory, and the T: directory has not yet been created or assigned, you will receive an error message.
- Feel free to add comments to your scripts. If you insert a semicolon at the end of a command line, anything to the right of the semicolon is ignored by AmigaDOS. It just appears in the script as a comment to remind you of what you are trying to accomplish. For instance:

ASSIGN T: RAM:T ; set up T directory for scripts

is a quick reminder of why you inserted that command and what you wanted it to do. Be sure to include a space after the command line and before the semicolon.

If you make a mistake, you may see one of the following error messages:

Unknown command <command>

This occurs when you have entered a command that is unrecognizable.

<command> failed returncode 20

This occurs if you've entered the command's arguments incorrectly. Use the WHY command to get a better explanation of the error. (Type WHY at the prompt after the error appears.)

If an error appears, use your text editor (ED, MEMacs or a word processor which can save ASCII files) to correct the line containing that command. Depending on where the error occurred, you may have to reboot with a different Workbench disk before being able to access the error. If you have a hard disk, you may have to reboot with a floppy disk.

You can use a special Shell function to try to pinpoint the exact location of an error. Entering SET ECHO ON within a script will cause all command lines to be ECHOed to the screen as they are executed. The error message will be printed after the system tries to execute the incorrect command. To disable SET ECHO, enter SET ECHO OFF or delete the SET ECHO ON line.

Common Additions to the Startup Files

This section describes some simple additions that can be made to your startup files. Be sure to make any changes on a backup copy of the Workbench disk.

To automatically open a Shell window:

Add the following lines to the User-startup file:

```
cd SYS: ; You could also make this cd RAM:  
NewShell CON:0/0/640/200/AmigaShell/CLOSE
```

You could substitute any name you like for AmigaShell.

To set up additional paths and logical device names:

If you always boot with a specific floppy disk in your external disk drive, you can add additional paths and logical device names to your User-startup file. For instance, if you always boot with the Extras2.0 disk in DF2:, you could add the following line:

```
PATH Extras2.0: ADD
```

If you always boot with a C source, Includes, and Lib disk in DF2:, you might find it convenient to assign logical names to these directories. If your scripts always refer to these logical names, you will only have to change these ASSIGN statements when you change your system or configuration. For instance, if the disk is named MySrc, you could add the following lines:

```
ASSIGN SRC: MySrc:  
ASSIGN INCLUDE: MySrc:Include  
ASSIGN LIB: MySrc:Lib
```

To make additional commands resident:

All of the C directory commands, the RexxC commands, DiskCopy, Format, ,MEmacs and More can be made resident, provided you have the necessary RAM. (If the pure protection bit of a command is set, you can make the command resident.) Resident commands are extremely fast, reduce memory usage

when multitasking, and generally make using the Shell more convenient, especially on a floppy-based system. These commands should be added to your User-startup file.

```
RESIDENT C:DELETE  
RESIDENT C:ED  
RESIDENT SYS:Utilities/More
```

When the Startup-sequence checks for the existence of the User-startup file (lines 33 to 35 in the previous example), these commands will be executed.

For more information on working around the limitations of a single floppy-based system, please read the next section, "For Single Floppy Disk Systems."

For Single Floppy Disk Systems

If your system has only one floppy drive and no hard disk, you must be prepared for a certain amount of disk swapping in the course of your work. AmigaDOS is a disk-based system and needs to load many of its commands from the Workbench disk before it can execute them.

If you need a file on another disk, such as a data disk containing text files, you will have to swap disks frequently. You need to insert the Workbench disk so that the Amiga can read the command information, then insert the data disk so the Amiga can execute the command. For instance, if you want to rename a file on your data disk, the system will need to read the RENAME program from the Workbench disk, then you'll have to insert the data disk so that it can actually rename the file.

There are a few AmigaDOS commands, such as **RESIDENT** and **ASSIGN**, that you can use to minimize the amount of disk swapping you have to do. These commands are explained in the following sections.

You can also minimize disk swapping by using the Ram Disk. This is explained in "The Ram Disk" section on page 7-72.

Making Commands Resident

A number of important AmigaDOS commands are Internal and do not need to be loaded from disk. While you cannot make commands Internal, you can make other commands resident so that you do not need to have the Workbench disk in the drive when you use them. Making commands resident essentially copies the program into the Amiga's free memory. When the command is invoked, the program information is used in memory instead of being read from disk. This is also faster than loading the command from disk.

Making commands resident uses memory. Ideally, you should only make resident the commands that you use most often. Otherwise, you may be taking valuable RAM away from other programs. To determine approximately how much memory will be used if you make a command resident, use the **LIST** command. For instance,

```
1> LIST C:COPY
Directory "Sys:C" on Monday 25-Jun-90
copy      3552 --p-rwed 20-Jun-90 17:22:02
```

The size of the file is shown to the right of the filename. In this case, the **COPY** command is 3552 bytes. If it is made resident, it will consume approximately 3552 bytes of RAM.

On a system with minimal RAM (512K), you may want to make DELETE, INFO and ASSIGN resident. If you have additional memory, you might also want to make ED, MAKEDIR, RENAME, DISKCOPY and FORMAT resident. When making commands resident, think about the commands you use most often. Some commands, such as ADDBUFFERS, BINDDRIVERS, and LOADWB should not be made resident as they are usually executed only during the startup sequence.

To see the correct format and the available options of the RESIDENT command, refer to the "Resident" section of Chapter 8.

Using ASSIGN's PATH Option

Another way to reduce the frequency of disk swaps is with the PATH option of the ASSIGN command. Normally, AmigaDOS will look on the original boot disk for any commands, device drivers, libraries, and other system software it needs.

If another disk is in the disk drive, you will get a requester asking for the original boot disk. This requester will appear even if the disk currently in the drive contains the file the system needs. The PATH option of the ASSIGN command allows you to direct AmigaDOS to look for the files it needs on any disk inserted in the designated drive.

To use the PATH option, you should add the following commands to your User-startup script:

```
ASSIGN C: DF0:C PATH  
ASSIGN L: DF0:L PATH  
ASSIGN LIBS: DF0:Libs PATH  
ASSIGN DEVS: DF0:Devs PATH  
ASSIGN FONTS: DF0:Fonts PATH
```

This makes using several different disks more convenient. You can also copy system directories onto application disks that may require them. You will no longer need to keep reinserting the original boot volume.

For the correct format and other options of the ASSIGN command, see the "Assign" section of Chapter 8.

Making Room on Your Workbench Disk

If you try to add programs to your Workbench disk, such as fonts or printer drivers, you will find that the disk is very full. You can try to eliminate some files from the Workbench disk to make room for other files you may need.

This involves deleting system software from your Workbench disk. If you decide to try this, be sure you are working with a copy of the Workbench disk, not the original. The original, unchanged Workbench disk should always be kept safe in case you need to restore a deleted file.



Any deletion of system software results in some limitation of your Amiga's capabilities. Depending on which Amiga features you use, you may not notice the limitation. However, some application may eventually attempt to use a file you have deleted, leading to an unexpected requester or error. It may not be obvious that the problem is the missing file.

If this happens, try the same operation with your original Workbench disk. If the error does not appear, you will know that it was caused by the absence of the deleted file(s).

Be sure to document any changes you make to your system disks. You may also want to add a statement in the disk's User-startup file to remind you that you are working with a non-standard Workbench.

When deleting files from your Workbench2.0 disk, you should start with the least crucial files first, such as the Clock and Exchange programs in the Utilities directory. Also, if you do not change your Preferences settings very often, you can delete the individual Preferences editors in the Prefs directory. By moving these programs, and their icons, to a different disk you can delete approximately 200K of data from the Workbench2.0 disk.



Do not delete Display, More, the Env-Archive subdirectory of Prefs, or the entire Prefs directory as other applications may call upon these programs or directories. If the program cannot be found, the application may fail inexplicably.

If you only use one of the AmigaDOS editors, ED, EDIT, or MEMacs, you can delete the editors you do not use. Do not delete all three editors, however. You should always have at least one editor on the disk in case you need to modify your User-startup file or perform some other quick editing function. MEMacs is the largest editor taking up over 50K. ED takes up approximately 24K, while EDIT only consumes 14K.

If after deleting these programs you still need additional room, you may want to delete the programs that control the Amiga's speech capability: DEVS:Narrator.device, LIBS:Translator.library, and L:Speak-handler. However, this is not recommended, as it may not be obvious which application

programs use these files, and you could experience unexplained software failures. Deleting these programs will make approximately 75K of disk space available.

Finally, if you absolutely must have more space, you can delete the files pertaining to the AREXX programming language: REXXC:, System/RexxMast, System/RexxMast.info, LIBS:rexsyslib.library, and LIBS:rexsupport.library. This will make almost 50K of disk space available. Again, this is not recommended as many application programs may call upon these files.

Files that you should not delete under any circumstances are listed below.

- C:IPrefs
- DEVS:MountList
- DEVS:parallel.device
- DEVS:printer.device
- DEVS:serial.device
- LIBS:asl.library
- LIBS:Commodities.library
- LIBS:Diskfont.library
- LIBS:Iffparse.library
- S:Startup-sequence
- S:Shell-startup
- L:Port-handler

Be careful when deciding what to eliminate. Don't delete any more than you have to in order to get the new material to fit. If you don't know the purpose of a file, leave it alone.

You may end up with several Workbench disks, each customized to allow the Amiga to work optimally with different programs.

The Ram Disk

RAM: and the Ram Disk are the same device. RAM: is the device name, while Ram Disk is the volume name shown under the disk icon.

RAM:, represented on the Workbench screen by the Ram Disk icon, is an area of the Amiga's internal memory that is set up as a file storage device like a disk. Files, directories, and (available memory permitting) entire floppy disks can be copied to RAM: for temporary storage.

The size of RAM: is dynamic. It is never any larger than necessary to hold its contents. Therefore, it is always 100% full. Its maximum size is limited by the amount of free memory.

The primary advantage of RAM: is speed. Since it is electronic, rather than mechanical, storage and retrieval are almost instantaneous. The disadvantage of RAM: is that data stored in RAM: does not survive when the computer is powered down or rebooted.

Applications commonly use RAM: for the storage of temporary files created during the use of the program or backup files created when the program is exited. This way they do not force the user to have a floppy disk available. RAM: can also be used for the storage of experimental script files, as a destination for testing command output, and whenever the creation of a file on an actual disk would be too slow, risky or inconvenient.

Advantages for Floppy Disk Systems

For floppy disk systems, RAM: is particularly useful when you are doing something that requires repeated disk accesses to a group of related files. If you can load the group of files into

RAM:, work with them individually while they are in RAM:, then copy them back to the floppy disk when the operation is finished, you only have to access the floppy disk twice. All the other file operations would take place internally in RAM:. This would speed up the process considerably.

For instance, suppose you have a directory called Brushes which contains two dozen IFF files, and you need to modify each file with a graphic program called Paint. If you worked solely from your floppy disk, you would have to run Paint, load each brush individually, change it, and save it back to disk. If you had to do this for each of the twenty-four files, it could take a considerable amount of time.

However, if you copied the IFF files into RAM:, you could run Paint, load each brush directly from RAM:, change it, then save it back to RAM:. After you were finished with the files, you could copy the entire group back to the floppy disk. The following commands illustrate how this could be accomplished through the Shell:

```
1> COPY DF0:Brushes TO RAM:Brushes ALL
```

This both creates a Brushes directory in RAM: and copies the contents of the Brushes directory on DF0: to the new directory in RAM:.

```
1> RUN PAINT
```

This command loads the Paint program. You could then load each IFF file into Paint, change it, then save it. The only difference is instead of specifying the path to the files as DF0:Brushes, you would use RAM:Brushes.

After you have changed and saved all of the IFF files, you could copy them back to your floppy disk.

```
1> COPY RAM:Brushes TO DF0:Brushes ALL
```

On a single-floppy system, RAM: can be used to reduce the amount of disk swapping required for floppy-to-floppy

transfers. By making your temporary, incremental saves to RAM:, you can keep the Workbench or program disk in the drive until you need to save data to disk.

Be careful when using RAM: for storage of important files. If the Amiga loses power, has a software failure, or you reboot, everything stored in RAM: will be lost. Be sure when working with RAM: to regularly back up any important files on a floppy disk.

NOTE: You cannot copy a disk to RAM: by dragging the source disk icon over the Ram Disk icon. To copy a disk to RAM:, you should open the Ram Disk icon, and drag the floppy disk icon into the Ram Disk window. This will create a drawer with the name and contents of the floppy disk.

The Recoverable Ram Disk

AmigaDOS also provides a recoverable Ram Disk, usually mounted as RAD:. The contents of RAD: will survive a reboot and most software failures, making it a safer place for work files. (Data in RAD: will still be lost when the computer is turned off.)

Unlike RAM:, RAD: is not automatically created. However, there is an entry for RAD: supplied in the MountList file in DEVS:. To activate a recoverable Ram Disk, you simply add the following line to your User-startup file:

```
MOUNT RAD:
```

Unlike RAM:, the size of RAD: is not dynamic. It is fixed in the MountList. You can change its size by entering a different value in the HighCyl parameter of the RAD: MountList entry.

A HighCyl entry of 79 results in a RAD: with the same capacity as a normal 880K floppy disk.

Properly set up, RAD: can also make working with a floppy disk system much faster. On a 1MB Amiga with no hard drive, a small RAD: can be used to hold your S: directory and some common AmigaDOS commands. If you have more than 2MB of RAM, you may want to create a floppy-size RAD:. The Workbench disk can then be copied to RAD: for a recoverable Workbench-in-RAM. You could then reboot from RAD: instead of from the Workbench2.0 disk. Instructions for setting up sample RAD: devices are given below.

You can also set up multiple RAD: devices of different sizes by copying the RAD: MountList entry and changing the name and unit number.

For a 1MB Amiga

A small RAD: is not suitable for rebooting because it cannot hold all of the commands and handlers required during startup. However, it is useful for freeing up DF0: since it can hold some commonly-used commands, the S directory, and the Expansion directory.

1. *Use an editor to change the Devs/MountList file. In the RAD: entry, set the HighCyl value to 14, and add this line:*

```
BootPri = -129
```

This specifies a non-bootable RAD:.

2. *Remove the following line from the Startup-sequence:*

```
path ram: c: sys:utilities sys:rexxc sys:system s: sys:prefs sys:wbstartup add
```


3. Insert the following lines in your User-startup file:

```
failat 30
assign >NIL: RAD: exists
if warn
    echo "Mounting RAD:..."
    mount RAD:
if not exists
    echo "Setting up RamDrive..."
    relabel RAD: RamDrive
    resident c:copy
    resident c:makedir
    mkdir rad:c rad:utilities rad:expansion rad:system rad:s
    copy c:assign c:info c:loadwb c:copy c:dir c:ed c:iprefs to rad:c quiet
    copy c:makedir c:rename to rad:c quiet
    copy s: rad:s all quiet
    copy sys:system/cli sys:system/format rad:system quiet
    copy sys:system/diskcopy rad:system quiet
    copy sys:expansion rad:expansion quiet
    copy sys:utilities/more rad:utilities quiet
endif
endif
failat 10
assign s: rad:s
path reset rad:c rad:system rad:utilities sys:utilities sys:rexxc sys:system
path sys:prefs sys:wbstartup add
echo "Done"
```

You may wish to modify the COPY lines of the above example to place some different commands in RAD:. The directories are required, and some of the commands, such as Format and DiskCopy, are strongly recommended.

4. Add these two lines to the Startup-sequence before the LOADWB line:

```
path C: RAD: add
assign C: RAD:c
```

For Amigas with more than 2MB

A rebootable RAD: is extremely useful if you have a megabyte of RAM to spare and no hard drive. On powerup, this will mount an 880K RAD: and copy your boot disk to it. This takes about 30 seconds and is not repeated when you reboot. All system directories are assigned to RAD: so that the rest of the Startup-sequence is executed very quickly. DF0: is left free for use as a work drive.

To do this, follow the steps below, then turn your Amiga off for at least 30 seconds. Turn it back on, and boot with your modified Workbench2.0 disk. When the DiskCopy of DF0: to RAD: is complete, a requester will flash twice.

1. Insert the following lines into your User-startup file:

```
failat 30
assign >NIL: RAD: exists
if warn
    echo "Mounting RAD:..."
    mount RAD:
    if not exists RAD:c
        echo "Copying DF0: to RAD:"
        sys:system/diskcopy <NIL: DF0: to RAD: name "RamWB"
    endif
endif
echo "Transferring control to RAD: . . . ."
assign C: RAD:c
assign S: RAD:s
assign L: RAD:l
assign LIBS: RAD:libs
assign DEVS: RAD:devs
assign FONTS: RAD:fonts
assign SYS: RAD:
echo "Done"
```

Other Workbench Directories

In addition to the AmigaDOS commands explained in the “AmigaDOS Reference” chapter, there are many other files and directories on your Workbench disk. This section discusses the S:, L:, DEVS:, FONTS:, and LIBS: directories. Some of the files are new, while others have been revised with the release of Version 2.0. The standard contents of these directories may change as resources are added, changed, or removed.

It is not necessary for most Amiga users to have a detailed understanding of the contents of these directories. However, you may run into problems if you should inadvertently delete or rename a file in a directory or fail to copy something to the appropriate directory.

Each of these directories is automatically assigned to the SYS: volume, which is the Workbench disk or hard disk partition that the Amiga boots from. You may ASSIGN these directories to different volumes if you wish.

For instance, you can assign FONTS: to a particular disk, such as FontDisk:. Many applications automatically look for the fonts that they need in the FONTS: directory regardless of which disk it is assigned to. If the application can't find FONTS: you may get an error message or have a problem using the program.

The S: Directory

The S: directory is generally reserved for scripts. In addition to the Startup-sequence and Shell-startup files explained earlier in this chapter, the S: directory also contains several other scripts which are explained below.

ED-Startup

This file contains a series of ED commands used to configure the ED text editor, assigning the default function key options. It can be edited to customize the key assignments.

Other files containing ED commands may be stored in S: for use by the WITH keyword of ED, allowing ED command files to perform custom editing operations.

HDBackup.config

This file is used with the HDBackup hard disk backup and restore utility.

SPat, DPat

These scripts add pattern matching to commands which don't naturally support it. When run with a command, SPat and DPat use the LIST command to create temporary script files in the T: directory, then execute the scripts. SPat and DPat can be used within command aliases.

SPat adds pattern matching to single-argument commands. For example, to use the More utility to display all the files in the S: directory that begin with the letter "s", you would enter:

```
1> SPat More S:s#?
```

A script would be generated, similar to this:

```
more "s:Shell-startup"  
more "s:SPat"  
more "s:Startup-sequence"
```

SPat would then execute the script, invoking More three times to display the files.

DPat adds pattern matching to double-argument commands. After DPat and the command name, enter the two arguments, separated by a space, using the wildcards required to produce the desired matches.

PCD

Similar to the CD command, PCD also remembers the last directory. For example, typing:

```
1> PCD RAM:
1> PCD
```

will change the current directory to RAM:, then return you to the starting directory.

The DEVS: Directory

The DEVS: directory contains several files and subdirectories pertaining to the different devices that can be used with the Amiga. DEVS: contains several .device files, some of which correspond to actual physical devices, such as peripherals attached to the Amiga's ports. The .device files and their functions are listed below:

clipboard.device	Controls writing and reading clips to CLIPS:.
narrator.device	Controls access to the speech synthesizer.
parallel.device	Controls access to the parallel port.
printer.device	Controls access to the printer device.
serial.device	Controls access to the serial port.

This section does not explore the .device files. They are explained in other reference works, such as the *ROM Kernel Manuals* published by Addison-Wesley.

This section does cover the MountList, which many users need to become familiar with when they install expansion devices in their Amiga, and the Keymaps and Printers subdirectories.

MountList

The MountList file contains the descriptions of devices that are to be mounted with the AmigaDOS MOUNT command. You may need a MountList entry for a device, handler, or file system. When you add a new device to your Amiga system, such as a hard disk or even some external disk drives, you must make the Amiga aware of the existence of the device. Some devices automount using the expansion directory. For others, you must use the MOUNT command. The MOUNT command must read a MountList entry in order to determine the characteristics of the device.

Several sample MountList entries are already included in the MountList file in the DEVS: directory. Some of these can be used without changes, but it is always a good idea to verify the file to make sure it accurately corresponds with your device.

A MountList entry consists of a number of keywords describing the device, handler, or file system, as well as values for those keywords. Some keywords may only apply to a file system or a handler. If a keyword is omitted, a default value is used. You should always check the default value in case it is not appropriate for whatever you are mounting.

There are certain rules for creating a MountList entry:

- Each entry in the MountList must start with the name of the device.
- Keywords are followed by an equals sign (=).
- Keywords must be separated by a semicolon or by placing them on a separate line.
- Comments are allowed in standard C style (i.e., comments start with /* and end with */).
- Each entry must end with the # symbol, on a line of its own.

The keywords supported by the Mountlist are shown in the tables on pages 7-82 and 7-83.

Mountlist Entries

Keyword	Function
Handler =	A handler entry (i.e., Handler = L:Speak-Handler).
EHandler =	An environment handler entry.
FileSystem =	A file system entry (i.e., FileSystem = L:FastFileSystem).
Device =	A device entry (i.e., Device = ramdrive.device).
Priority =	The priority of the process; 5 is good for handlers, 10 for file systems.
Unit =	The unit number of the device.
Flags =	Flags for OpenDevice (usually 0).
Surfaces =	The number of surfaces.
BlocksPerTrack =	The number of blocks per track.
Reserved =	The number of blocks reserved for the boot block; should be 2.
PreAlloc =	The number of blocks reserved from the end of a partition; used with a few hard drives that store information in the last few blocks of a drive. This is usually set to 0 and probably will not need to be changed. Please refer to the documentation packaged with your hard drive and hard drive controller.
Interleave =	Interleave value; varies with the device.
LowCyl =	Starting cylinder to use.
HighCyl =	Ending cylinder to use.
Stacksize =	Amount of stack allocated to the process.
Buffers =	Number of initial cache buffers.
BufMemType =	Memory type used for buffers; (0 and 1 = Any, 2 and 3 = Chip, 4 and 5 = Fast).

Mountlist Entries

Keyword	Function
Mount =	If a positive value, MOUNT loads the device or handler immediately rather than waiting for first access.
MaxTransfer =	The maximum number of bytes transferred; used with the FastFileSystem.
Mask =	Address Mask to specify memory range that DMA transfers can use; used with the FastFileSystem.
GlobVec =	A global vector for the process; -1 is no Global Vector (for C and assembler programs), 0 sets up a private GV; if the keyword is absent, the shared Global Vector is used.
Startup =	A string passed to the device, handler, or filesystem on startup as a BPTR to a BSTR.
BootPri =	A value which sets the boot priority of a bootable and mountable device. This value can range from -129 to 127. By convention, -129 indicates that the device is not bootable and is not automatically mounted.
DosType =	Indicates the type of file system. If the FastFileSystem is used, DosType should be set to 0x444F5301. Otherwise, the DosType should be 0x444F5300. Or, you could simply omit it altogether. It is possible that other values may be used in the future.
Baud =	Serial device baud rate.
Control =	Serial device word length, parity, and stop bits.

Sample MountList entries are included in the MountList file. Usually if you need to create a new MountList, you will be given instructions in the documentation that accompanies the device you are mounting. There are also several MountList examples in this chapter accompanying the descriptions of the various handlers in the L: directory.

Keymaps

Keymaps is a subdirectory of DEVS: (Devs/Keymaps). International keymaps are available in the Devs/Keymaps directory of the Extras2.0 disk. If you have a hard disk, they are in the DEVS:Keymaps directory.

Available Keymaps	
Keymap	Keyboard it represents
cdn	French-Canadian
chl	Swiss-French
ch2	Swiss-German
d	German
dk	Danish
e	Spanish
f	French
gb	British
i	Italian
is	Icelandic
n	Norwegian
s	Swedish
usa0	For V1.0 programs
usa2	Dvorak

To use an international keymap:

1. *Copy the keymap file to the DEVS:Keymaps directory. For example:*

```
1> COPY Extras2.0:Devs/Keymaps/D TO DEVS:Keymaps
```

2. *Use the SetMap program (in the System drawer) to inform the system of the change.*

1> System/SETMAP D

If you want to use a different keymap on a regular basis, copy the file to DEVS:Keymaps and insert the SetMap assignment in your User-startup file.

The standard United States keymap (usa1) is built into ROM.

Printers

If you have a floppy disk system, the Printers subdirectory of the Workbench2.0 disk is empty. In order to use a printer with your Amiga, you must copy the appropriate printer driver from the Extras2.0:Devs/Printers subdirectory to your Workbench2.0:Devs/Printers subdirectory. This is explained in Chapter 3, "Preferences."

If you have a hard disk system, all the available printer drivers will be in your DEVS:Printers directory. For complete specifications on the available drivers, see Appendix B.



The L: Directory

This directory contains the **device handlers**, software **modules** that act as intermediate stages between AmigaDOS and the devices used by the Amiga. However, most handlers are treated as if they are actual physical devices and are referred to by their device name. For instance, SPEAK: represents the Speak-Handler which provides speech output for the Amiga.

Handlers must be named in the MountList entry for their respective devices. Handlers generally are not called or manipulated directly by users, but by programs. New handlers may be supplied with some devices or programs and should be added to the L: directory.

AUX: is mounted during the standard Startup-sequence.

Aux-Handler

The Aux-Handler provides unbuffered serial input and output. It is essentially a console handler that uses the serial port rather than the Amiga screen and keyboard.

The MountList entry is:

```
AUX:
  Handler = L:Aux-handler
  Stacksize = 1000
  Priority = 5
#
```

A sample entry is already in the MountList file.

You can use Aux-Handler to use another terminal with your computer. For example:

```
1> NEWSHELL AUX:
```

Pipe-Handler

The Pipe-Handler is an I/O mechanism used to provide input/output communication between programs. It essentially creates an interprocess communication channel. When the information is directed to PIPE: up to 4K of data are buffered before the writing process is blocked. After you write to a PIPE:, another process can read the data.

PIPE: is mounted during the standard Startup-sequence.

The MountList entry is:

```
PIPE:
  Handler = L:Pipe-handler
  Stacksize = 6000
  Priority = 5
  GlobVec = -1
#
```

PIPE: may be used from other programs, like a word processor (as a filename during a save operation) or a terminal program (as a capture buffer filename). You can use any pipe-name you wish. PIPE: uses a 4K internal buffer per name, but its optimal

situation is one in which one program is reading while one program is writing. When using PIPE:, the source and destination processes must be distinct (not the same process).

The buffer is transparent. This means that data written, no matter how little it is, is immediately available to be read by the other process.

The PIPE: device can be useful when you're using two programs and want to transfer large amounts of data from one (write) to the other (read) without using a temporary file in RAM: or on disk. Assuming the application does not attempt to read the file non-sequentially, you simply specify PIPE:<name> and it looks like an ordinary file to the application.

You can also copy information from one PIPE: to another. For example:

```
Shell window 1: COPY Hugefile PIPE:a
Shell window 2: COPY PIPE:a PIPE:b
Shell window 3: COPY PIPE:b PIPE:c
Shell window 4: COPY PIPE:c PIPE:d
Shell window 5: COPY PIPE:d PIPE:e
Shell window 6: TYPE PIPE:e ;Hugefile will be TYPed
```

Port-Handler

The Port-Handler is the AmigaDOS interface for the SER:, PAR:, and PRT: devices.

Speak-Handler

The Speak-Handler provides speech output for the Amiga. With SPEAK: you can literally have the Amiga say the contents of a file.

SPEAK: is mounted during the standard Startup-sequence.

The MountList entry is:

```
SPEAK:
    Handler = L:Speak-handler
    Stacksize = 4000
    Priority = 5
    GlobVec = -1
#
```

In addition to the MountList entry, SPEAK: also requires the narrator.device and translator.library. They must be in DEVS: and LIBS: respectively. (These files are included on Workbench2.0 in the appropriate directories.)

The format for using SPEAK: is:

SPEAK:OPT/K

After the OPT keyword, the following options may be used:

The options must be separated by a slash (/) and there should be no space between the colon and OPT.

p###	Pitch (where ### is from 65-320)
s###	Speed (where ### is from 30-400)
m	Male voice
f	Female voice
r	Robot inflection
n	Natural inflection
o0	Do not allow these options in the input stream.
o1	Allow these options in the input stream.
a0	Turn off direct phoneme mode.
a1	Turn on direct phoneme mode (do not use translator.library).
d0	Break up sentences on punctuation alone.
d1	Break up sentences on punctuation, Return, and line feed.

SPEAK: may be used from other programs, like a word processor (as a filename during a save operation) or a terminal program (as a capture buffer filename) to get spoken output.

For example, to listen to the contents of your Startup-sequence file, type:

```
1> COPY S:Startup-sequence to SPEAK:OPT/f/s160
```

The contents of the Startup-sequence will be read in a female voice at a moderate speed.

The FONTS: Directory

The FONTS: directory contains the information for all of the different styles of fonts available to the Amiga. For each font, there is a subdirectory and a .font file.

For instance, for the Emerald font, there is an Emerald directory and an Emerald.font file. The font directory contains files for the different point sizes that are available. The Emerald directory contains two files: 17 and 20. The files contain the data needed for the 17 point Emerald font and the 20 point Emerald font. The Emerald.font file contains the list of point sizes and any available styles, such as bold, italics, etc., for the font.

Many word processor or desktop publishing programs contain additional fonts that you should copy to your FONTS: directory. Whenever you add a new font to FONTS: you should run the FixFonts program to create the .font file for the new addition.

The Topaz font is the default font used by the Amiga. In addition to existing in the FONTS: directory, it is built into ROM. Even if you deleted the entire FONTS: directory, the Amiga would still be able to display text.

The LIBS: Directory

LIBS: contains a variety of software routines and math functions commonly-used by the operating system and applications.

LIBS: Files	
.library File	Function
asl.library	File and font requester modules
commodities.library	Modules used by Commodities Exchange programs
diskfont.library	Library modules for finding and loading font files
iffparse.library	Modules to parse IFF files
mathieeedoubbas.library	Double-precision IEEE math routine modules for basic functions (addition, subtraction, etc.)
mathieeedoubtrans.library	Double-precision IEEE math routine modules for transcendental functions (sine, cosine, etc.)
mathieeesingtrans.library	Fast single-precision IEEE math routine modules
mathtrans.library	FFP transcendental function math routine modules
rexksupport.library	Modules used by AREXX
rexksyslib.library	Main AREXX modules
translator.library	Speech-synthesis modules for translating English text into phonemes suitable for narrator.device
version.library	Contains current software version and revision information

Chapter 8. AmigaDOS Reference

This chapter gives complete specifications of all the AmigaDOS 2.0 commands. All the AmigaDOS commands have been improved, and several new commands have been added. Many commands are now Internal (built into the Shell) for speed, convenience, and reduced memory usage.

This chapter includes:

- command conventions, an explanation of the symbols and abbreviations used in the command descriptions
- specifications for each command, including the Workbench and Preferences programs
- a table of error messages

A Quick Reference list of the AmigaDOS commands is included at the back of this manual.

Command Conventions

When you invoke an AmigaDOS command, you usually do more than type the command name at a Shell prompt. Many commands require arguments or support options that send the Amiga additional information as to what you want to do. For example, if you type:

1> DIR Utilities

you are telling the Amiga to generate a list of files and subdirectories stored in the Utilities directory. In this command line, Utilities is an argument. However, if you typed:

```
1> DIR Utilities FILES
```

only a list of the files in the Utilities directory would be shown; subdirectories would not be listed. Here, FILES is an option.

In the "Command Specifications" section, the AmigaDOS commands are explained following a standard outline:

<i>Format</i>	All the arguments and options accepted by a command.
<i>Template</i>	A built-in reminder of the command's format. The template is embedded in the program's code. If you type a command followed by a question mark (DIR ?), the template will appear on the screen.
<i>Purpose</i>	A short explanation of the command's function.
<i>Path</i>	The directory where the command is normally stored. For most commands this will be the C: directory. The exceptions are the Internal commands which are copied into memory and the Workbench programs.
<i>Specification</i>	A description of the command and all of its arguments.
<i>Examples</i>	When examples are given, the command and any screen output are indented from the main text. A generic 1> prompt indicates what should be typed at the Shell prompt. All command names and arguments are capitalized for clarity. Case does not matter when entering commands. To execute the command line, you must press Return.

Remember, commands and arguments should be separated by spaces. (It does not have to be just one space; multiple spaces are acceptable.) No other punctuation should be used unless it is called for in the syntax of the specific command.

Format

In Format listings, arguments are enclosed in different kinds of brackets to indicate the type of argument. The brackets are not to be typed as part of the command.

- < > Angle brackets enclose arguments that must be provided. For instance, <filename> means that you must enter the appropriate filename in that position. Unless square brackets surround the argument (see below), the argument is required. The command will not work unless it is specified.
- [] Square brackets enclose arguments and keywords that are optional. They will be accepted by the command but are not required.
- { } Braces enclose items that can be given once or repeated any number of times. For example, {<args>} means that several items may be given for this argument.
- | A vertical bar is used to separate options of which you can choose only one. For example, [OPT R|S|RS] means that you can choose the R option, the S option, or both (RS) options.

The format for the COPY command is shown below:

```
COPY [FROM] {<name|pattern>} [TO] <name|pattern> [ALL]  
[QUIET] [BUF|BUFFER = <n>] [CLONE] [DATE] [NOPRO] [COM]
```

The [FROM] keyword is optional. If it is not specified, the command reads the filename or pattern to copy by its position on the command line.

The {<name|pattern>} argument must be provided. You must substitute either a filename or pattern. The braces indicate that more than one argument can be given.

The [TO] keyword is optional. If it is not specified, the command reads the filename or device to copy to by its position on the command line.

The <name|pattern> argument must be provided. You can only specify one destination.

The [ALL], [QUIET], [CLONE], [DATE], [NOPRO], and [COM] arguments are optional.

The [BUF|BUFFER = <n>] argument is optional. If given, you can use either BUF or BUFFER and a numerical argument. For instance, both BUF = 5 and BUFFER = 5 are acceptable.

Template

The Template is more condensed than the Format and is built into the system. If you type a question mark (?) after a command, the Template will appear to remind you of the proper syntax.

In Template listings, arguments are separated by commas and followed by a slash (/) and a capital letter which indicates the type of argument. The slash/letter combinations, which are not to be typed as part of the command, are explained below:

argument/A	The argument must always be given.
option/K	The option's keyword must be used if the argument is given.
option/S	The option works as a switch . You must type the name of the option in order to specify that option. Most options are switches.
value/N	The argument is numeric.
argument/M	Multiple arguments are accepted. This is the Template equivalent of braces. The /M replaces the previous multiple-comma method of indicating how many elements the command could operate upon. There is no limit on the number of possible arguments. However, the multiple arguments must be given before any other argument or option.
string/F	The string must be the final argument on the command line. The remainder of the command line is taken as the desired string. Quotation marks are not needed around the string, even if it contains spaces. If you type quotation marks, they will be passed to the command. If you specify the keyword, you can pass leading and trailing spaces.

=

An equals sign indicates that two different forms of the keyword are equivalent. Either will be accepted. The equals sign is not typed as part of the command.

The Template for the COPY command is shown below:

FROM/A/M, TO/A, ALL/S, QUIET/S, BUF = BUFFER/K/N, CLONE/S, DATES/S,
NOPRO/S, COM/S

FROM/A/M indicates that the argument must be given and multiple arguments are acceptable.

TO/A indicates that the argument must be given.

ALL/S, QUIET/S, CLONE/S, DATES/S, NOPRO/S, COM/S indicate that the keywords act as switches. If the keyword is present in the line, the option will be used.

BUF = BUFFER/K/N indicates that a numerical (/N) argument is optional (/K). Both BUF and BUFFER are acceptable keywords.

Command Specifications

A

ADDBUFFERS

Format: ADDBUFFERS <drive> [<n>]

Template: DRIVE/A,BUFFERS/N

Purpose: To command the file system to add cache buffers.

Path: C:ADDBUFFERS

Specification:

ADDBUFFERS adds <n> buffers to the list of buffers available for <drive>. Allocating additional buffers makes disk access significantly faster. However, each additional buffer reduces free memory by approximately 500 bytes. The default buffer allocation is 5 for floppy drives and usually 30 for hard disks.

The number of buffers you should add depends on the amount of extra memory available. There is no fixed upper limit, but adding too many buffers can actually reduce overall system performance by taking RAM away from other system functions. If a negative number is specified, that many buffers are subtracted from the current allocation. The minimum number of buffers is one; however, using only one is not recommended.

Thirty buffers are generally recommended for a floppy drive in a 512K system. The optimal number for a hard disk depends on the type and size of your drive. Usually you should use the default value recommended by the HDToolbox program explained in Chapter 6. (This value can be displayed by selecting the Advanced Options gadget on the Partitioning screen.) As a general rule, you can use 30 to 50 buffers for every megabyte of RAM in your system.

If only the <drive> argument is specified, ADDBUFFERS displays the number of buffers currently allocated for that drive.

A buffer is a temporary storage area in memory.

Examples:

```
1> ADDBUFFERS DF1: 25
DF1: has 30 buffers
```

Adds 25 buffers to drive DF1:.

```
1> ADDBUFFERS DF0:
DF0: has 20 buffers
```

Displays the number of buffers currently allocated to drive DF0:.

ADDMONITOR

Format: ADDMONITOR NUM = %d NAME = %s

Template: NUM/N/A,NAME/A,HBSTRT/K,HBSTOP/K,
HSSTRT/K,HSSTOP/K,VBSTRT/K,VBSTOP/K,
VSSTRT/K,VSSTOP/K,MINROW/K,MINCOL/K,
TOTROWS/K,TOTCOLS/K,BEAMCONO/K

Purpose: To inform the Amiga that a non-RGB style monitor has been added to your system.

Path: SYS:System/AddMonitor

Specification:

ADDMONITOR must be run if you have attached an A2024 or Multiscan monitor or a monitor that is different from your country's video standard (PAL for NTSC countries, and vice versa). The acceptable values for NUM and NAME are listed below:

	NUM =	NAME =
For an NTSC monitor	1	NTSC
For a PAL monitor	2	PAL
For a Multiscan monitor	3	Multiscan
For an A2024 monitor	4	A2024

The additional options control special hardware features of the Amiga and are primarily of use to hardware developers. Certain graphics hardware may require these options in order to function correctly. If so, the options should be explained in the documentation accompanying the hardware.

After using AddMonitor, you must use the ScreenMode editor to select the new display mode. To have the system recognize your monitor upon booting, drag the appropriate icon from the MonitorStore drawer to the Monitors drawer.

Examples:

If you've attached a Multiscan monitor, type:

```
1> ADDMONITOR NUM = 3 NAME = Multiscan
```

If you've attached an A2024 monitor, type:

```
1> ADDMONITOR NUM = 4 NAME = A2024
```

You must then use the ScreenMode Preferences editor to select the appropriate display mode for your monitor.

ALIAS

Format: ALIAS [<name>] [<string>]

Template: NAME,STRING/F

Purpose: To set or display command aliases.

Path: Internal

Specification:

ALIAS permits you to create aliases, or alternative names, for AmigaDOS commands. Using an alias is like replacing a sentence with a single word. With ALIAS, you can abbreviate frequently used commands or replace a standard command name with a different name.

When AmigaDOS encounters <name>, it replaces it with the defined <string>, integrates the result with the rest of the command line, and attempts to interpret and execute the resulting line as an AmigaDOS command. So <name> is the

alias (whatever you want to call the command), and <string> is the command to be substituted for the alias.

An alias must be at the beginning of the command line, and you can specify arguments on the command line after the alias. However, you cannot use an alias for a series of command arguments. For instance, you cannot create a script using the LFORMAT option of the LIST command by creating an alias to represent the LFORMAT argument.

You can substitute a filename or other instruction within an alias by placing square brackets ([]) in the <string>. Any argument typed after the alias will be inserted at the brackets.

ALIAS <name> displays the <string> for that alias. ALIAS alone lists all current aliases.

Aliases are local to the Shell in which they are defined. If you create another Shell with the NEWSHELL command, it will share the same aliases as its parent Shell. However, if you create another Shell with the Execute Command menu item, it will not recognize aliases created in your original Shell. To create a global alias that will be recognized by all Shells, insert the alias in the Shell-startup file.

To remove an ALIAS, use the UNALIAS command.

Examples:

```
1> ALIAS d1 DIR DF1:
```

Typing d1 results in a directory of the contents of the disk in DF1:, just as if you had typed DIR DF1:.

```
1> ALIAS hex TYPE [ ] HEX NUMBER
```

creates an alias called hex that displays the contents of a specified file in hexadecimal format. The brackets indicate where the filename will be inserted. If you then typed:

```
1> hex Myfile
```

the contents of Myfile would be displayed in hexadecimal format with line numbers.

ASK

Format: ASK <prompt>

Template: PROMPT/A

Purpose: To obtain user input when executing a script file.

Path: Internal

Specification:

ASK is used in scripts to write the <prompt> to the current window, then wait for keyboard input. Valid responses are Y (yes), N (no), and Return (no). If Y is pressed, ASK sets the condition flag to 5 (WARN). If N is pressed, the condition flag is set to 0. To check the response, an IF statement can be used.

If the <prompt> contains spaces, it must be enclosed in quotation marks.

Example:

Assume a script contained the following commands:

```
ASK Continue?  
IF WARN  
    ECHO Yes  
ELSE  
    ECHO No  
ENDIF
```

When the ASK command is reached, Continue? will appear on the screen. If Y is pressed, Yes will be displayed on the screen. If N is pressed, No will be displayed.

See also: IF, ELSE, ENDIF, WARN

ASSIGN

Format: ASSIGN [<name>:{dir}] [LIST] [EXISTS]
[DISMOUNT] [DEFER] [PATH] [ADD] [REMOVE]
[VOLS] [DIRS] [DEVICES]

Template: NAME,TARGET/M,LIST/S,EXISTS/S,
DISMOUNT/S,DEFER/S,PATH/S,ADD/S,
REMOVE/S,VOLS/S,DIRS/S,DEVICES/S

Purpose: To control assignment of logical device names to file system directories.

Path: C:ASSIGN

Specification:

ASSIGN allows directories to be referenced via short, convenient logical device names rather than their usual names or complete paths. ASSIGN gives an alternative directory name, much as ALIAS permits alternative command names. The ASSIGN command can create assignments, remove assignments, or list some or all current assignments.

If the <name> and {dir} arguments are given, ASSIGN will assign the given name to the specified directory. Each time the assigned logical device name is referred to, AmigaDOS will access the specified directory. If the <name> given is already assigned to a directory, the new directory will replace the previous directory. (Always be sure to include a colon after the <name> argument.)

If only the <name> argument is given, any existing ASSIGN of a directory to that logical device will be cancelled.

You can assign several logical device names to the same directory by using multiple ASSIGN commands.

You can assign one logical device name to several directories by specifying each directory after the <name> argument or by using the ADD option. When the ADD option is specified,

any existing directory assigned to <name> is not cancelled. Instead, the newly specified directory is added to the assign list, and the system will search for both directories when <name> is encountered. If the original directory is not available, ASSIGN will be satisfied with the newly added directory.

To delete a name from the assign list, use the REMOVE option. If no arguments are given with ASSIGN, or if the LIST keyword is used, a list of all current assignments will be displayed. If the VOLS, DIRS, or DEVICES switch is specified, ASSIGN will limit the display to volumes, directories, or devices, respectively.

When the EXIST keyword is given along with a logical device name, AmigaDOS will search the ASSIGN list for that name and display the volume and directory assigned to that device. If the device name is not found, the condition flag is set to 5 (WARN). This is commonly used in scripts.

Normally, when the {dir} argument is given, AmigaDOS immediately looks for that directory. If the ASSIGN commands are part of the startup-sequence, the directories need to be present on a mounted disk during the boot procedure. If an assigned directory cannot be found, a requester appears asking for the volume containing that directory. However, two new options, DEFER and PATH, will wait until the directory is actually needed before searching for it.

The DEFER option creates a "late-binding" ASSIGN. This assign only takes effect when the assigned object is first referenced, rather than when the assignment is made. This eliminates the need to insert disks during the boot procedure that contain the directories that are assigned during the startup-sequence. When the DEFER option is used, the disk containing the assigned directory is not needed until the object is actually called upon. The assignment remains in force until explicitly changed.

It is not necessary for the assigned name to retain the name of the directory nor for it to be uppercase.

For example, if you assign FONTS: to DF0:Fonts with the DEFER option, the system will associate FONTS: with whatever disk is in DF0: at the time FONTS: is called. If you have a Workbench disk in DF0: at the time the FONTS: directory is needed, the system will associate FONTS: with that particular Workbench disk. If you later remove that Workbench disk and insert another disk containing a Fonts directory, the system will specifically request the original Workbench disk the next time FONTS: is needed.

The PATH option creates a “non-binding” ASSIGN. A non-binding ASSIGN acts like a DEFERred ASSIGN except that it is reevaluated each time the assigned name is referenced. This prevents the system from expecting a particular volume in order to use a particular directory (such as the situation described in the example above). For instance, if you assign FONTS: to DF0:Fonts with the PATH option, any disk in DF0: will be searched when FONTS: is referenced. As long as the disk contains a Fonts directory, it will satisfy the ASSIGN. You cannot assign multiple directories with the PATH option.

The PATH option is especially useful to users with floppy disk systems as it eliminates the need to reinsert the original Workbench disk used to boot the system. As long as the drive you have assigned with the PATH option contains a disk with the assigned directory name, the system will use that disk.



The DISMOUNT option (called REMOVE in V1.3) disconnects a volume or device from the list of mounted devices. It does not free up resources; it merely removes the name from the list. There is no way to cancel a DISMOUNT without rebooting. DISMOUNT is primarily for use during software development. Careless use of this option may cause a software failure.

Examples:

```
1> ASSIGN FONTS: MyFonts:Fontdir
```

assigns the system FONTS: directory to Fontdir on MyFonts:.

```
1> ASSIGN LIST
```

Volumes:

Ram Disk [Mounted]

Workbench2.0 [Mounted]

MyFonts [Mounted]

Directories:

REXX Workbench2.0:

CLIPS Ram Disk:Clipboards

ENV Ram Disk:Env

T Ram Disk:T

ENVARC Workbench2.0:Prefs/Env-Archive

SYS Workbench2.0:

C Workbench2.0:C

S Workbench2.0:S

L Workbench2.0:L

FONTS MyFonts:Fontdir

DEVS Workbench2.0:Devs

LIBS Workbench2.0:Libs

Devices:

PIPE AUX SPEAK RAM CON

RAW PAR SER PRT DF0 DF1

shows a list of all current assignments.

```
1> ASSIGN FONTS: EXISTS
```

```
FONTS MyFonts:FontDir
```

is an inquiry into the assignment of FONTS:. AmigaDOS responds by showing that FONTS: is assigned to the FontDir directory of the MyFonts volume.

```
1> ASSIGN LIBS: SYS:Libs BigAssem:Libs PDAssem:Libs
```

is a multiple-directory assignment that creates a search path containing three Libs directories. These directories will be searched in sequence each time LIBS: is invoked.

```
1> ASSIGN DEVS: DISMOUNT
```

removes the DEVS: assignment from the system.

```
1> ASSIGN WorkDisk: DF0: DEFER
1> ASSIGN WorkDisk: EXISTS
WorkDisk <DF0:>
```

sets up a late-binding assignment of the logical device WorkDisk:. The disk does not have to be inserted in DF0: until the first time you refer to the name WorkDisk:. Notice that ASSIGN shows DF0: enclosed in angle brackets to indicate that it is DEFERred. After the first reference to WorkDisk:, the volume name of the disk that was in DF0: at the time will replace <DF0:>.

```
1> ASSIGN C: DF0:C PATH
1> ASSIGN C: EXISTS
C          [DF0:C]
```

will reference the C directory of whatever disk is in DF0: at the time a command is searched for. Notice that ASSIGN shows DF0:C in square brackets to indicate that is a non-binding ASSIGN.

```
1> ASSIGN LIBS: ZCad:Libs ADD
```

adds ZCad:Libs to the list of directories assigned as LIBS:.

```
1> ASSIGN LIBS: ZCad:Libs REMOVE
```

removes ZCad:Libs from the list of directories assign as LIBS:.

AUTOPOINT

Format: AUTOPOINT [CX_PRIORITY = <n>]

Template: CX_PRIORITY/K/N

Purpose: To automatically select any window the pointer is over.

Path: Extras2.0:Tools/Commodities/AutoPoint

Specification:

When AUTOPOINT is run, any window that the pointer is over is automatically selected. You do not need to click the selection button to activate it.

The `CX_PRIORITY = <n>` argument sets the priority of AutoPoint in relation to all the other Commodity Exchange programs. (This is the same as entering a `CX_PRIORITY = <n>` Tool Type in the icon's Information window.) All the Commodity Exchange programs are set to a default priority of 0. If you specify an `<n>` value higher than 0, AutoPoint will take priority over any other Commodity Exchange program.

To exit AutoPoint when it has been started from a Shell, type `Ctrl-E` or use the `BREAK` command.

Example:

```
1> AUTOPOINT
```

starts the AutoPoint program.

AVAIL

Format: AVAIL [CHIP|FAST|TOTAL] [FLUSH]

Template: CHIP/S,FAST/S,TOTAL/S,FLUSH/S

Purpose: To report the amount of Chip and Fast memory available.

Path: C:AVAIL

Specification:

AVAIL gives a summary of the system RAM, both Chip and Fast. For each memory type, AVAIL reports the total amount, how much is available, how much is currently in use, and the largest contiguous memory block not yet allocated.

By using the `CHIP`, `FAST` and/or `TOTAL` options, you can have AVAIL display only the number of free bytes of Chip, Fast or total RAM available, instead of the complete summary. This value can be used for comparisons in scripts.

The Amiga uses two different types of RAM. Chip RAM is used for graphics and sound data. Fast RAM is general purpose RAM used by all types of programs.

The FLUSH option causes all unused libraries and device modules to be expunged from memory.

Examples:

1> AVAIL

Type	Available	In-Use	Maximum	Largest
chip	233592	282272	515864	76792
fast	341384	182896	524280	197360
total	574976	465168	1040144	197360

1> AVAIL CHIP

233592

BINDDRIVERS

Format: BINDDRIVERS

Template: (none)

Purpose: To bind device drivers to hardware.

Path: C:BINDDRIVERS

Specification:

BINDDRIVERS is used to load and run device drivers for add-on hardware that is configured by the expansion library. These device drivers must be in the SYS:Expansion directory for BINDDRIVERS to find them.

BINDDRIVERS is normally placed in the Startup-sequence file. If drivers for expansion hardware are in the Expansion directory, you must have a BINDDRIVERS command in your Startup-sequence or the hardware will not be configured when the system is booted.

BINDMONITOR

B

Format: BINDMONITOR <MONITORID>
<MONITORNAME>

Template: MONITORID/A,MONITORNAME/A

Purpose: To assign names to the different display modes.

Path: SYS:System/BindMonitor

Specification:

BINDMONITOR assigns names to the different display modes currently supported by the graphics library. The acceptable arguments match the Tool Types of the ModeNames icon.

≡ Acceptable BINDMONITOR Arguments ≡

0×00000 Lores	0×00004 Lores-Interlaced
0×08000 Hires	0×08004 Hires-Interlaced
0×08020 SuperHires	0×08024 SuperHires-Interlaced
0×11000 NTSC:Lores	0×11004 NTSC:Lores-Interlaced
0×19000 NTSC:Hires	0×19004 NTSC:Hires-Interlaced
0×19020 NTSC:SuperHires	0×19024 NTSC:SuperHires-Interlaced
0×21000 PAL:Lores	0×21004 PAL:Lores-Interlaced
0×29000 PAL:Hires	0×29004 PAL:Hires-Interlaced
0×29020 PAL:SuperHires	0×29024 PAL:SuperHires-Interlaced
0×31004 VGA-ExtraLores	0×31005 VGA-ExtraLores-Interlaced
0×39004 VGA-Lores	0×39005 VGA-Lores-Interlaced
0×39024 Productivity	0×39025 Productivity Interlaced
0×41000 A2024_10Hz	0×49000 A2024_15Hz

For instance, ROM recognizes 0×08000 as a 640 × 200 line display. However, BindMonitor links 0×08000 with the name Hires. The names associated with the display modes appear in the Choose Display Mode gadget of the ScreenMode editor.

Example:

```
1> BINDMONITOR 0×08000 MyDisplay
```

MyDisplay would appear in the Choose Display Mode gadget of the ScreenMode editor instead of Hires.

BLANKER

Format: BLANKER [SECONDS = <n>]
[CX_POPKEY = <key (s)>]
[CX_POPUP = <yes|no>]
[CX_PRIORITY = <n>]

Template: SECONDS/K/N,CX_POPKEY/K,CX_POPUP/K,
CX_PRIORITY/K/N

Purpose: To cause the monitor screen to go blank if no input has been received within a specified period of time.

Path: Extras2.0:Tools/Commodities/Blanker

Specification:

BLANKER is a Commodity Exchange program that causes the screen to go blank if no mouse or keyboard input has been received in the specified number of seconds. The SECONDS = <n> argument allows you to specify the number of seconds that must pass. The acceptable range is from 1 to 9999. Default is 60 seconds.

A list of acceptable key combinations can be found on page 5-29.

CX_POPKEY = <key(s)> allows you to specify the hot key for the program. If more than one key is specified, be sure to enclose the entire argument in double-quotes (i.e., "CX_POPKEY = Shift F1").

CX_POPUP = no will prevent the Blanker window from opening. (By default the program window opens when the command is invoked.)

CX_PRIORITY = <n> sets the priority of Blanker in relation to all other Commodity Exchange programs. All the Commodity Exchange programs are set to a default priority of 0.

To kill Blanker when it is run through the Shell, press Ctrl-E.

Examples:

```
1> BLANKER SECONDS=45
```

The Blanker window will open, and 45 will be displayed inside its text gadget. If no mouse or keyboard input is received during a 45 second interval, the screen will go blank.

```
1> BLANKER CX_POPUP=no
```

The Blanker program will start. If no input is received within 60 seconds (the default), the screen will go blank. The Blanker window will not open.

BREAK

Format: BREAK <process> [ALL|C|D|E|F]

Template: PROCESS/A/N,ALL/S,C/S,D/S,E/S,F/S

Purpose: To set attention flags in the specified process.

Path: C: BREAK

Specification:

BREAK sets the specified attention flags in the <process> indicated. C sets the Ctrl-C flag, D sets the Ctrl-D flag, and so on. ALL sets all the flags from Ctrl-C to Ctrl-F. By default, AmigaDOS only sets the Ctrl-C flag.

The action of BREAK is identical to selecting the relevant process by clicking in its window and pressing the appropriate Ctrl-key combination[s].

Ctrl-C is used as the default for sending a BREAK signal to halt a process. A process that has been aborted this way will display ***BREAK in the Shell window. Ctrl-D is used to halt execution of a script file. Ctrl-E is used to exit Commodity Exchange programs. Ctrl-F is not currently used.

Use the STATUS command to display the current process numbers.

Examples:

1> BREAK 7

sets the Ctrl-C attention flag of process 7. This is identical to selecting process 7 and pressing Ctrl-C.

1> BREAK 5 D

sets the Ctrl-D attention flag of process 5.

See also: STATUS

CALCULATOR

Format: CALCULATOR

Template: (none)

Purpose: To provide an on-screen calculator.

Path: SYS:Utilities/Calculator

Specification:

CALCULATOR starts the Calculator program. You can cut-and-paste the output of the Calculator into any console window, like the Shell or ED.

To exit the program, select the window's close gadget.

Example:

1> CALCULATOR

CD

Format: CD [<dir|pattern>]

Template: DIR

Purpose: To set, change, or display the current directory.

Path: Internal

Specification:

CD with no arguments displays the name of the current directory. When a valid directory name is given, CD makes the named directory the current directory.

CD does not search through the disk for the specified directory. It expects it to be in the current directory. If it is not, you must give a complete path to the directory. If CD cannot find the specified directory in the current directory or in the given path, a Can't find <directory> error message is displayed.

If you want to move up a level in the filing hierarchy to the parent directory of the current directory, type CD followed by a space and a single slash (/). Moving to another directory in the parent can be done at the same time by including its name after the slash. If the current directory is a root directory, CD / will have no effect. Multiple slashes are allowed; each slash refers to an additional higher level. When using multiple slashes, leave no spaces between them.

To move directly to the root directory of the current device, use CD followed by a space and a colon.

CD also supports pattern matching. If more than one directory matches the given pattern, an error message is displayed.

Examples:

```
1> CD DF1:Work
```

sets the current directory to the Work directory on the disk in drive DF1:.

```
1> CD SYS:Com/Basic
```

makes the subdirectory Basic in the Com directory the current directory.

```
1> CD //
```

moves up two levels in the directory structure and makes SYS: the current directory.

```
1> CD SYS:Li#?
```

uses the #? pattern to match with the Libs directory.

CHANGETASKPRI

Format: CHANGETASKPRI <priority> [<process>]

Template: PRI = PRIORITY/A/N,PROCESS/K/N

Purpose: To change the priority of a currently-running process.

Path: C:CHANGETASKPRI

Specification:

Use the STATUS command to display the current process numbers.

Since the Amiga is multitasking, it uses priority numbers to determine the order in which current tasks should be serviced. Normally, most tasks have a priority of 0, and the time and instruction cycles of the CPU are divided equally among them. CHANGETASKPRI changes the priority of the specified Shell process. (If no process is specified, the current Shell process is assumed.) Any tasks started from <process> inherit its priority.

The range of acceptable values for <priority> is the integers from -128 to 127, with higher values yielding a higher priority (a greater proportion of CPU time is allocated). However, do not enter values above +10, or you may disrupt important system tasks. Too low a priority (less than 0) can result in a process taking unreasonably long to execute.

Example:

```
1> CHANGETASKPRI 4 Process 2
```

The priority of Process 2 is changed to 4. Any tasks started from this Shell will also have a priority of 4. They will have priority over any other user tasks created without using CHANGETASKPRI (those tasks will have a priority of 0).

See also: STATUS

CLOCK

Format: CLOCK [DIGITAL] [[LEFT] <n>] [[TOP <n>]
[[WIDTH] <n>] [[HEIGHT <n>] [24HOUR]
[SECONDS] [DATE]

Template: DIGITAL/S,LEFT/N,TOP/N,WIDTH/N,
HEIGHT/N,24HOUR/S,SECONDS/S,DATE/S

Purpose: To provide an on-screen clock.

Path: SYS:Utilities/Clock

Specification:

The DIGITAL option opens a digital clock.

The LEFT, TOP, WIDTH, and HEIGHT options allow you to specify the size and position of the clock. The keywords are optional; however, the clock understands numerical arguments by their position, as outlined below:

- 1st number The clock will open <n> pixels from the left edge of the screen.
- 2nd number The clock will open <n> pixels from the top of the screen.
- 3rd number The clock will be <n> pixels wide.
- 4th number The clock will be <n> pixels high.

For instance, if you only wanted to specify the width and height of the Clock, you would have to use the WIDTH and HEIGHT keywords. If you only typed two numbers, the clock would interpret them as the LEFT and TOP positions.

NOTE: WIDTH and HEIGHT are not available if you use the DIGITAL option. You cannot change the size of the digital clock, although you can specify its position.

The 24HOUR option opens the clock in 24-hour mode. If not specified, the clock opens in 12-hour mode.

If the SECONDS option is specified, the seconds are displayed.

If the DATE option is specified, the date is displayed.

Examples:

To open a clock that is 75 pixels from the left edge of the screen, 75 pixels from the top edge of the screen, 300 pixels wide and 100 pixels high, type:

```
1> CLOCK 75 75 300 100
```

To use the SECONDS, DATE and 24HOUR options, type:

```
1> CLOCK SECONDS DATE 24HOUR
```

To open a digital clock that is 320 pixels from the left edge of the screen and in the screen's title bar (0 pixels from the top), type:

```
1> CLOCK DIGITAL 320 0
```

CMD

Format: CMD <devicename> <filename> [OPT s|m|n]

Template: DEVICENAME/A,FILENAME/A,OPT/K

Purpose: To redirect printer output to a file.

Path: Extras2.0:Tools/CMD

Specification:

The <devicename> can be serial, parallel or printer, and should be the same device as specified in the Printer editor. <Filename> is the name of the file to which the redirected output should be sent.

The CMD options are as follows:

- s Skip any short initial write (usually a reset if redirecting a screen dump).
- m Intercept multiple files until a BREAK command or Ctrl-C is typed.
- n Notify user of progress (messages are displayed on the screen).

Example:

```
1> CMD parallel ram:cmd_file
```

Any output sent to the parallel port will be rerouted to a file in RAM: called cmd_file.

COLORS

Format: COLORS [<bitplanes> <screentype>]

Template: BITPLANES,SCREENTYPE

Purpose: To change the colors of the frontmost screen.

Path: Extras2.0:Tools/Colors

Specification:

COLORS lets you change the colors of the frontmost screen. By specifying values for the <bitplanes> and <screentype> options you can open a custom test screen. The acceptable values for <bitplanes> and <screentype> are listed below:

<bitplanes>	Specifies the depth of the test screen:
1	2 colors
2	4 colors
3	8 colors
4	16 colors
5	32 colors
<screentype>	Specifies the resolution of the test screen:
0	320 x 200 pixels
1	320 x 400 pixels
2	640 x 200 pixels
3	640 x 400 pixels

The value for <bitplanes> is restricted to 4 or less if the value for <screentype> is equal to either 2 or 3.

Example:

```
1> COLORS 3 2
```

A new custom screen will be opened, and it will display a window for the color program. The screen will have 8 colors and a 640 x 200 pixel (Hires) resolution.

CONCLIP

Format: CONCLIP [UNIT <n>] [OFF]

Template: UNIT/N,OFF/S

Path: C:CONCLIP

Purpose: To move data between the console.device, the clipboard.device and CON:.

Specification:

CONCLIP is called from the standard Startup-sequence. When it is run, the user can copy text from standard Shell windows by drag selecting text with the mouse, as explained in the "Copying and Pasting" section of Chapter 7. Once the text is highlighted, it can then be copied to the clipboard by pressing right Amiga-C. In addition, some other console.device windows may support the ability to drag select text, such as ED and MEmacs. The copied text can then be pasted into any application window which supports reading text from the clipboard, such as the Shell, ED, and MEmacs. To paste text, press right-Amiga V.

CONCLIP requires iffparse.library and the clipboard.device and opens the first time that you copy or paste text. Because of this, users with floppy-based systems may notice some delay as iffparse.library and the clipboard.device are loaded from disk (assuming that they have not already been loaded by some other application).

The UNIT option allows you to specify the clipboard.device unit number to use. You can specify any unit from 0 to 255. The default unit number is 0. This option is primarily for advanced users or programmers who may want to use different units for different data, such as one for text and another for graphics. You do not need to turn CONCLIP off to change the UNIT number. Simply, run the command from the Shell specifying the new unit number. The next time you copy and paste, that clipboard unit will be used.

The OFF option allows the more advanced user or programmer to turn off CONCLIP. When turned off, text is not copied to the clipboard, and pasting is transparently managed by the console.device. In general, there is not reason to turn CONCLIP off.

COPY

Format: COPY [FROM] {<name|pattern>} [TO]
<name|pattern> [ALL] [QUIET]
[BUF|BUFFER = <n>] [CLONE] [DATES]
[NOPRO] [COM] [NOREQ]

Template: FROM/A/M,TO/A,ALL/S,QUIET/S,
BUF = BUFFERS/K/N,CLONE/S,DATES/S,
NOPRO/S,COM/S,NOREQ/S

Purpose: To copy files or directories.

Path: C:COPY

Specification:

COPY copies the file or directory specified with the FROM argument to the file or directory specified by the TO argument. You can copy several items at once by giving more than one FROM argument; each argument should be separated by spaces. You can use pattern matching to copy or exclude items whose names share a common set of characters or symbols.

If a TO filename already exists, COPY overwrites the TO file with the FROM file. If you name a destination directory that does not exist, COPY will create a directory with that name. You can also use a pair of double quotes (""") to refer to the current directory when specifying a destination. (Do not put any spaces between the double quotes.)

If the FROM argument is a directory, only the directory's files will be copied; its subdirectories will not be copied. Use the ALL option to copy the complete directory, including its files, subdirectories, and the subdirectories' files. If you want to copy a directory and you want the copy to have the same name as the original, you must include the directory name in the TO argument.

COPY prints to the screen the name of each file as it is copied. This can be overridden by the QUIET option.

The BUF= option is used to set the number of 512-byte buffers used during the copy. (Default is 200 buffers, approximately 100K of RAM.) It is often useful to limit the number of buffers when copying to RAM:. BUF=0 uses a buffer the same size as the file to be copied.

Normally, copy gives the TO file the date and time the copy was made. Any comments attached to the original FROM file are ignored. The protection bits of the FROM file are copied to the TO file. Several options allow you to override these defaults:

- DATES The creation date of the FROM file is copied to the TO file.
- COM Any comment attached to the FROM file is copied to the TO file.
- NOPRO The protection bits of the FROM file are not copied to the TO file. The TO file will be given standard protection bits of r, w, e and d.
- CLONE The date, comments and protection bits of the FROM file are copied to the TO file.

Normally, COPY displays a requester if the COPY cannot continue for some reason. When the NOREQ option is given, all requesters are suppressed. This is useful in scripts and can prevent a COPY failure from stopping the script while it waits for a response. For instance, if a script calls for a certain file to be copied and the system cannot find that file, normally the script would display a requester and would wait until a response was given. With the NOREQ option, the COPY command would be aborted and the script would continue.

Examples:

```
1> COPY File1 TO :Work/File2
```

copies File1 in the current directory to File2 in the Work directory.

```
1> COPY ~(#?.info) TO DF1:Backup
```

copies all the files not ending in .info in the current directory to the Backup directory on the disk in DF1:. This is a convenient use of pattern matching to save storage space when icons are not necessary.

```
1> COPY Work:Test TO ""
```

copies the files in the Test directory on Work to the current directory; subdirectories in Test will not be copied.

```
1> COPY Work:Test TO DF0:Test ALL
```

copies all the files and any subdirectories of the Test directory on Work to the Test directory on DF0:. If a Test directory does not already exist on DF0:, AmigaDOS will create one.

```
1> COPY DF0: TO DF1: ALL QUIET
```

copies all files and directories on the disk in DF0: to DF1:, without displaying on the screen any file/directory names as they are copied. (This is quite slow in comparison to DiskCopy.)

CPU

C

Format: CPU [CACHE] [BURST] [NOBURST]
[DATACACHE] [DATABURST]
[NODATACACHE] [NODATABURST]
[INSTCACHE] [INSTBURST] [NOINSTCACHE]
[NOINSTBURST] [FASTROM] [NOFASTROM]
[NOMMUTEST] [CHECK 68010|68020|68030|
68881|68882|68851|MMU|FPU]

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,
NOMMUTEST/S,CHECK/K

Purpose: To set or clear the CPU caches, check for a particular processor, load the ROM image into fast, 32-bit memory, or set an illegal memory access handler which will output information over the serial port at 9600 baud if a task accesses page zero (lower 256 bytes) or memory above 16M.

Path: C:CPU

Specification:

CPU allows you to adjust various options of the microprocessor installed in your Amiga. CPU will also show the processor and options that are currently enabled.

NOTE: Many options only work with certain members of the 680XO processor family. The 68020 has a special type of memory known as instruction cache. When instruction cache is used, instructions are executed more quickly. The 68030 has two types of cache memory: instruction and data. If you have

Static Column Dynamic RAM (SCRAM) installed, you can also use a special access mode for both instruction and data cache, known as burst mode. This may further improve access speed in some cases. The CPU options, outlined below, specify the types of memory to be used. If mutually exclusive options are specified, the safest option is used.

CACHE	Turns on both data and instruction cache (only for 68030).
NOCACHE	Turns off data and instruction cache.
BURST	Turns on burst mode for both data and instructions (only for 68030 with SCRAM).
NOBURST	Turns off burst mode for data and instructions.
DATACACHE	Turns on data cache (only for 68030).
NODATACACHE	Turns off data cache.
DATABURST	Turns on burst mode for data (only for 68030 with SCRAM).
NODATABURST	Turns off burst mode for data.
INSTCACHE	Turns on instruction cache.
INSTBURST	Turns on burst mode for instructions (if SCRAM installed).
NOINSTCACHE	Turns off instruction cache.
NOINSTBURST	Turns off burst mode for instructions.
FASTROM	Copies data from ROM into 32-bit RAM, making access to this data significantly faster. CPU then write-protects the RAM area so that the data cannot be changed.
NOFASTROM	Turns off FASTROM.

NOMMUTEST Allows the MMU to be changed without checking to see if it is currently in use.

The **CHECK** option, when given with a keyword (68010, 68020, 68030, 68881, 68882, or 68851) checks for the presence of the keyword.

Examples:

```
1> CPU
System: 68030 68881 (INST: NoCache Burst) (DATA: Cache NoBurst)

1> CPU Burst Cache Check MMU
System: 68030 68881 (INST: Cache Burst) (DATA: Cache Burst)

1> CPU NoBurst DataCache NoInstCache
System: 68030 68881 (INST: NoCache NoBurst) (DATA: Cache NoBurst)

1> CPU Burst Cache FastROM
System: 68030 68881 FastROM (INST: Cache Burst) (DATA: Cache Burst)

1> CPU NoFastRom NoDataCache
System: 68030 68881 (INST: Cache Burst) (DATA: NoCache Burst)
```

DATE

Format: DATE [<day>] [<date>] [<time>]
[TO|VER <filename>]

Template: DAY,DATE,TIME,TO = VER/K

Purpose: To display or set the system date and/or time.

Path: C:DATE

Specification:

DATE with no argument displays the currently set system time and date, including the day of the week. Time is displayed using a 24-hour clock.

DATE <date> sets just the date. The format for <date> is DD-MMM-YY (day-month-year). The hyphens between the arguments are required. A leading zero in the date is not necessary. The first 3 letters of the month (in English) must be used, as well as the last two digits of the year.

C

D

If the date is already set, you can reset it by specifying a day name (this sets the date forward to that day of the week). You can also use tomorrow or yesterday as the <day> argument.

DATE <time> sets the time. The format for <time> is HH:MM:SS (hours:minutes:seconds). Seconds are optional.

If your Amiga does not have a battery backed-up hardware clock and you do not set the date, the system, upon booting, will set the date to the date of the most recently created file on the boot disk.

If you specify the TO or VER option, followed by a filename, the output of the DATE command will be sent to that file, overwriting any existing contents.

NOTE: Adjustments made with DATE only change the software clock. They will not survive past power-down. To set the battery backed-up hardware clock from the Shell, you must set the date then use SETCLOCK SAVE.

Examples:

```
1> DATE
```

displays the current date and time.

```
1> DATE 6-sep-82
```

sets the date to the 6th of September, 1982. The time is not reset.

```
1> DATE tomorrow
```

resets the date to one day ahead.

```
1> DATE TO Fred
```

sends the current date to the file Fred.

```
1> DATE 23:00
```

sets the current time to 11:00 p.m.

```
1> DATE 1-jan-02
```

sets the date to January 1st, 2002. (The earliest date you can set is January 1, 1978.)

DELETE

Format: DELETE {<name|pattern>} [ALL] [Q|QUIET]
[FORCE]

Template: FILE/M/A,ALL/S,QUIET/S,FORCE/S

Purpose: To delete files or directories.

Path: C:DELETE

Specification:

DELETE attempts to delete (erase) the specified file(s). If more than one file was specified, AmigaDOS continues to the next file in the list.

You can use pattern matching to delete files. The pattern may specify directory levels as well as filenames. All files that match the pattern are deleted. To abort a multiple-file DELETE, press Ctrl-C.

AmigaDOS does not request confirmation of deletions. An error in a pattern-matching DELETE can have severe consequences, as deleted files are unrecoverable. Be sure you understand pattern matching before you use this feature, and keep backups of important files.



If you try to delete a directory that contains files, you will receive a message stating that the directory could not be deleted as it is not empty. To override this, use the ALL option. DELETE ALL deletes the named directory, its subdirectories, and all files.

Filenames are displayed on the screen as they are deleted. To suppress the screen output, use the QUIET option.

If the d (deletable) protection bit of a file has been cleared, that file cannot be deleted unless the FORCE option is used.

Examples:

1> DELETE Old-file

deletes the Old-file file in the current directory.

1> DELETE Work/Prog1 Work/Prog2 Work

deletes the files Prog1 and Prog2 in the Work directory, and then deletes the Work directory (if there are no other files left in it).

1> DELETE T#?/#?(1|2)

deletes all the files that end in 1 or 2 in directories that start with T.

1> DELETE DF1:##? ALL FORCE

deletes all the files on DF1:, even those set as not deletable.

DIR

Format: DIR [<dir|pattern>] [OPT A|I|AI|D|F] [ALL]
[DIRS] [FILES] [INTER]

Template: DIR,OPT/K,ALL/S,DIRS/S,FILES/S,INTER/S

Purpose: To display a sorted list of the files in a directory.

Path: C:DIR

Specification:

DIR displays the file and directory names contained in the specified directory, or the current directory if no name is given. Directories are listed first, followed by an alphabetical list of the files in two columns. Pressing Ctrl-C aborts a directory listing.

The options are:

ALL	Displays all subdirectories and their files.
DIRS	Displays only directories.
FILES	Displays only files.
INTER	Enters an interactive listing mode.

NOTE: The ALL, DIRS, FILES and INTER keywords supersede the OPT A, D, F and I options, respectively. The older keywords are retained for compatibility with earlier versions of AmigaDOS. Do not use OPT with the full keywords—ALL, DIRS, FILES, or INTER.

The interactive listing mode stops after each name and displays a question mark at which you can enter commands. The acceptable responses are shown below:

Return	Displays the next name on the list.
E	Enters a directory; the files in that directory will be displayed.
B	Goes back one directory level.
DEL or DELETE	Deletes a file or empty directory.
T	Types the contents of a file.
C or COMMAND	Allows you to enter additional AmigaDOS commands.
Q	Quits interactive editing.
?	Displays a list of the available interactive-mode commands.

DEL does not refer to the Del key; type the letters D, E, then L.

The COMMAND option allows almost any AmigaDOS command to be executed during the interactive directory list. When you want to issue a command, type C (or COM) at the question mark prompt. DIR will ask you for the command. Type the desired command, then press Return. The command will be executed and DIR will continue. You can also combine the C and the command on one line, by putting the command in quotes following the C.

For instance, C "type prefs.info hex" is equivalent to pressing Q to exit interactive listing mode and return to a regular Shell prompt, and typing:

```
1> TYPE Prefs.info HEX
```



The Prefs.info file would be typed to the screen in hexadecimal format.

It is dangerous to format a disk from the DIR interactive mode, as the format will take place *immediately*, without any confirmation requesters appearing. Also, starting another interactive DIR from interactive mode will result in garbled output.

Examples:

1> DIR Workbench2.0:

displays a list of the directories and files on the Workbench2.0 disk.

1> DIR MyDisk: #?.memo

displays all the directories and files on MyDisk that end in .memo.

1> DIR Extras2.0: ALL

displays the complete contents of the Extras2.0 disk — all directories, all subdirectories and all files.

1> DIR Workbench2.0: DIRS

displays only the directories on Workbench2.0.

1> DIR Workbench2.0: INTER

provides an interactive list of the contents of Workbench2.0.

DISKCHANGE

Format: DISKCHANGE <device>

Template: DRIVE/A

Purpose: To inform the Amiga that you have changed a disk in a disk drive.

Path: C:DISKCHANGE

Specification:

The DISKCHANGE command is only necessary when you are using 5.25 inch floppy disk drives or removable media drives without automatic diskchange hardware. Whenever you change the disk or cartridge of such a drive, you must use DISKCHANGE to inform the system of the switch.

DISKCHANGE can also be used if you edit a disk icon image and wish to see the new icon on the Workbench screen immediately. This is the only way to display an altered hard disk icon without rebooting.

An altered floppy disk icon can be displayed by removing the disk from the drive and reinserting it.

Example:

If a requester appears and asks you to insert a new disk into your 5.25 inch drive, known as DF2:, you must insert the disk, then type:

```
1> DISKCHANGE DF2:
```

AmigaDOS will then recognize the new disk, and you can proceed.

DISKCOPY

Format: DISKCOPY [FROM] <disk> TO <disk>
[NOVERIFY] [MULTI] [NAME <name>]

Template: DISK/A,TO/A,DISK/A,NOVERIFY/S,
MULTI/S,NAME/S

Purpose: To copy the contents of one disk to another.

Path: SYS:System/DiskCopy

Specification:

The DISKCOPY command copies the entire contents of one volume to another. The FROM keyword does not have to be specified. However, the TO keyword must be given for DISKCOPY to work.

The <disk> argument can be either the volume name or drive name, such as Workbench2.0 or DF0:.

Normally during a diskcopy, the Amiga copies and verifies each cylinder of data. The NOVERIFY option allows you to skip the verification process, making the copy faster.

The MULTI option loads the data on the source disk into memory, allowing you to make multiple copies without having to read the data from the source disk each time.

By default, the destination disk will have the same name as the source disk. If you specify the NAME option, you can give the destination disk a different name from the source disk.

Examples:

1> DISKCOPY DF0: to DF2:

copies the contents of the disk in drive DF0: to the disk in drive DF2: overwriting the contents of the disk in drive DF2:

1> DISKCOPY DF0: to DF2: NOVERIFY NAME NewDisk

copies the contents of the disk in drive DF0: to the disk in drive DF2: and gives the disk in drive DF2: the name NewDisk. The disk will not be verified as it is copied.

DISKDOCTOR

Format: DISKDOCTOR <drive>

Template: DRIVE/A

Purpose: To attempt to repair a corrupted disk.

Path: C:DISKDOCTOR

Specification:

DISKDOCTOR attempts to repair a corrupted disk enough to allow you to retrieve files from it and copy them onto a good disk. If AmigaDOS has detected a corrupted disk, it displays a

requester stating that the disk could not be validated or that it has a read/write error. By using DISKDOCTOR, you can try to restore the file structure of the disk.

You can use DISKDOCTOR on both the standard file system and the FastFileSystem. However, to use DISKDOCTOR with the FastFileSystem, *you must make sure that the DosType keyword in the MountList is set to 0x444F5301*. Do not use DISKDOCTOR on a FastFileSystem partition if the DosType keyword is not set correctly.

DISKDOCTOR versions of 1.3.5 or earlier do not work with FFS floppies.



Before running DISKDOCTOR, it is a good idea to copy all files from the disk, as DISKDOCTOR will write to the corrupted disk. This can prevent the use of other disk-repair utilities. After running DISKDOCTOR, you should copy the restored files to a new disk, then reformat the corrupted disk.

DISKDOCTOR checks for enough memory before starting operations and changes the boot block to type DOS.

It may be necessary to run DISKDOCTOR several times before a disk is usable once again. If DISKDOCTOR was not able to read the root block of the disk, the disk will be renamed Lazarus.

Example:

If you receive a message stating that Volume Workbench is not validated or Error validating disk/Disk is unreadable, you can use DISKDOCTOR to retrieve the disk's files. For instance, if the corrupted disk is in DF1:, type:

```
1> DISKDOCTOR DF1:
```

AmigaDOS will ask you to insert the disk to be corrected and press Return. DISKDOCTOR then reads each cylinder of the disk. If it finds an error, it displays Hard error Track <xx>, Surface <xx>. As each file and directory is replaced, the filename is displayed on the screen. When DISKDOCTOR is finished, it displays Now copy files required to a new disk and reformat this disk.

If a hard error is found, there may be actual physical damage to the disk. If, after reformatting, the disk still shows problems, it should be discarded.

DISPLAY

Format: DISPLAY {<filename>|FROM <filelist>}
[OPT mlpbenv] [t = <n>]

Template: FILENAME/A/M, FROM/K, OPT/K, T/N

Purpose: To display graphics saved in IFF ILBM format.

Path: SYS:Utilities/Display

Specification:

DISPLAY displays graphics saved using the IFF ILBM format. You can type a series of files on the command line, and they will be shown in the order given. You can also create a script containing a list of all the IFF files you'd like to display and use the FROM <filelist> argument.

The options are listed below. Remember, the OPT keyword must be used.

- | | |
|---|--|
| m | Clicking the selection button displays the next file in the filelist; clicking the menu button displays the previous file. |
| l | Instead of exiting after the last picture, Display will return to the first file and start again. |

- b Pictures stay on their own unactivated screen behind the Workbench screen. This is useful when printing pictures while doing something else.
- p Prints each file that is displayed. You can also press Ctrl-P while the file is on the screen.
- a Pictures that are larger than the display area will scroll automatically when the pointer is moved to the edge of the screen.
- e This option tells Display to treat a 6-bitplane image as Extra Halfbrite. This is for users who may be using an early HAM paint package that does not save a CAMG chunk. Normally if there is no CAMG, Display will treat the image as a HAM picture.
- n Borders will not be transparent when genlocked.
- v Pictures will be displayed with full-video display clip. This means that the picture will fill the maximum possible position on the right edge of the screen, going a little beyond the Overscan settings in Preferences. However, when using this option, the screen cannot be dragged sideways, and Display cannot center the picture.

A CAMG chunk is part of an IFF file that describes in which viewmode the picture should be displayed.

D

The `t=<n>` argument specifies the number of seconds the IFF file will be displayed. This allows for automatic advancing through files.

Examples:

```
1> DISPLAY file1 file2 file3
```

displays the files in the order given. To advance from one file to the next, press Ctrl-C.

1> DISPLAY from Scriptlist

displays the files listed in the Scriptlist file. Pressing Ctrl-C will advance to the next file.

1> DISPLAY from Scriptlist OPT mp

displays the files listed in the Scriptlist file. Clicking the selection button advances to the next file in the list. Clicking the menu button displays the previous file. Each file is printed as it is displayed.

1> DISPLAY from Scriptlist OPT t=5

displays each file in the Scriptlist file for five seconds.

ECHO

Format: ECHO [<string>] [NOLINE] [FIRST <n>]
[LEN <n>] [TO<device|file>]

Template: /M,NOLINE/S,FIRST/K/N,LEN/K/N,TO/K

Purpose: To display a string.

Path: Internal

Specification:

ECHO writes the specified string to the current output window or device, usually the screen. By using the TO option, you can send the string to a device or file. When the string contains spaces, the whole string must be enclosed in double quotes. (ECHO is commonly used in scripts.)

When the NOLINE option is specified, ECHO does not automatically move the cursor to the next line after printing the string.

The FIRST and LEN options allow the echoing of a substring. FIRST <n> indicates the character position to begin the echo; LEN <n> indicates the number of characters of the substring to echo, beginning with the first character. If the FIRST option is omitted and only the LEN keyword is given, the substring printed will consist of the rightmost <n> characters of the

main string. For instance, if your string is 20 characters long and you specify LEN 4, the 17th, 18th, 19th, and 20th characters of the string will be echoed.

Examples:

```
1> ECHO "hello out there!"  
hello out there!
```

```
1> ECHO "hello out there!" NOLINE FIRST 0 LEN 5  
hello1>
```

E

ED

Format: ED [FROM] <filename> [SIZE <n>] [WITH]
[WINDOW] [TABS] [WIDTH] [HEIGHT]

Template: FROM/A,SIZE/N,WITH/K,WINDOW/K,
TABS/N,WIDTH=COLS/N,HEIGHT=ROWS/N

Purpose: To edit text files (a screen editor).

Path: C:ED

Specification:

See Chapter 9, "Editors."

EDIT

Format: EDIT [FROM] <filename> [[TO] <filename>]
[WITH <filename>] [VER <filename>]
[[OPT P <lines>|W <chars>]]
[PREVIOUS <lines>|WIDTH <chars>]]

Template: FROM/A,TO,WITH/K,VER/K,OPT/K,
WIDTH/N,PREVIOUS/N

Purpose: To edit text files by processing the source file sequentially (a line editor).

Path: C:EDIT

Specification:

See Chapter 9, "Editors."

ELSE

Format: ELSE

Template: (none)

Purpose: To specify an alternative for an IF statement in a script file.

Path: Internal

Specification:

ELSE is used in an IF block of a script to specify an alternative action in case the IF condition is not true. If the IF condition is not true, execution of the script will jump from the IF line to the line after ELSE; all intervening commands will be skipped. If the IF condition is true, the commands immediately following the IF statement are executed up to the ELSE. Then, execution skips to the ENDIF statement that concludes the IF block.

Example:

Assume a script, call Display, contained the following block:

```
IF exists <name>
  TYPE <name> OPT n
ELSE
  ECHO "<name> is not in this directory"
ENDIF
```

To execute this script, you could type:

```
1> EXECUTE Display work/prg2
```

If the work/prg2 file can be found in the current directory, the TYPE <name> OPT n command will be executed. The work/prg2 file will be displayed on the screen with line numbers.

If the work/prg2 file cannot be found in the current directory, the script will skip ahead to the ECHO "<name> is not in this directory" command. The message work/prg2 is not in this directory will be displayed in the Shell window.

See also: IF, ENDIF, EXECUTE

ENDCLI

Format: ENDCLI

Template: (none)

Purpose: To end a Shell process.

Path: Internal

Specification:

ENDCLI ends a Shell process.

See also: ENDSHELL

ENDIF

Format: ENDIF

Template: (none)

Purpose: To terminate an IF block in a script file.

Path: Internal

Specification:

ENDIF is used in scripts at the end of an IF block. If the IF condition is not true, or if the true condition commands were executed and an ELSE has been encountered, the execution of the script will skip to the next ENDIF command. Every IF statement must be terminated by an ENDIF.

The ENDIF applies to the most recent IF or ELSE command.

See also: IF, ELSE

ENDSHELL

Format: ENDSHELL

Template: (none)

Purpose: To end a Shell process.

Path: Internal

Specification:

*ENDCLI also closes a
Shell window.*

ENDSHELL ends a Shell process.

ENDSHELL should only be used when the Workbench is loaded or another Shell is running. If you have quit the Workbench and you close your only Shell, you will be unable to communicate with the Amiga. Your only recourse will be to reboot.

The Shell window may not close if any processes that were launched from the Shell are still running. Even though the window stays open, the Shell will not accept new input. You must terminate those processes before the window will close. For instance, if you opened an editor from the Shell, the Shell window will not close until you exit the editor.

ENDSKIP

Format: ENDSKIP

Template: (none)

Purpose: To terminate a SKIP block in a script file.

Path: Internal

Specification:

ENDSKIP is used in scripts to terminate the execution of a SKIP block. (A SKIP block allows you to jump over intervening commands if a certain condition is met.) When an ENDSKIP is encountered, execution of the script resumes at the line following the ENDSKIP. The condition flag is set to 5 (WARN).

See also: SKIP

EVAL

Format: EVAL <value1> [<operation>] [<value2>] [TO
<file>] [LFORMAT = <string>]

Template: VALUE1/A,OP,VALUE2/M,TO/K,LFORMAT/K

Purpose: To evaluate simple expressions.

Path: C:EVAL

Specification:

EVAL is used to evaluate and print the answer of an integer expression. The fractional portion of input values and final results, if any, is truncated (cut off).

<Value1> and <value2> may be in decimal, hexadecimal, or octal numbers. Decimal numbers are the default. Hexadecimal numbers are indicated by either a leading 0x or #X. Octal numbers are indicated by either a leading 0 or a leading #. Alphabetical characters are indicated by a leading single quote (').

You can specify multiple arguments for <value2>.

The output format defaults to decimal; however, you can use the LFORMAT keyword to select another format. The LFORMAT keyword specifies the formatting string used to print the answer. You may use %X (hexadecimal), %O (octal), %N (decimal), or %C (character). The %X and %O options require a number of digits specification (i.e., %X8 gives 8 digits of hex output). When using the LFORMAT keyword, you can specify that a new line should be printed by including a *N in your string.

The supported operations and their corresponding symbols are shown in the table below:

EVAL Operations	
Operation	Symbol
addition	+
subtraction	-
multiplication	*
division	/
modulo	mod
AND	&
OR	
NOT	~
left shift	<<
right shift	>>
negation	-
exclusive OR	xor
bitwise equivalence	eqv

EVAL can be used in scripts to act as a counter for loops. In that case, the TO option, which sends the output of EVAL to a file, is very useful.

Parentheses may be used in the expressions.

Examples:

```
1> EVAL 64 / 8 + 2
```

```
10
```

```
1> EVAL 0x4f / 010 LFORMAT = "The answer is %X4*N"
```

```
The answer is 0009
```

```
1>
```

This divides hexadecimal 4f (79) by octal 10 (8), yielding 0009, the integer portion of the decimal answer 9.875. (The 1> prompt would have appeared immediately after the 0009 if *N had not been specified in the LFORMAT string.)

Assume you were using the following script, called Loop:

```
.Key loop/a
; demo a loop using eval and skip
.Bra {
.Ket }
ECHO >ENV:Loop {loop}
LAB start
ECHO "Loop #" noline
TYPE ENV:Loop
EVAL <ENV:Loop >NIL: to=T:Qwe{$$} value2=1 op=- ?
TYPE >ENV:Loop T:Qwe{$$}
IF val $loop GT 0
SKIP start back
ENDIF
ECHO "done"
```

If you were to type:

```
1> EXECUTE Loop 5
Loop #5
Loop #4
Loop #3
Loop #2
Loop #1
done
```

The first ECHO command sends the number given as the loop argument, entered as an argument of the EXECUTE command, to the ENV:Loop file.

The second ECHO command coupled with the TYPE command, displays Loop # followed by the number given as the loop argument. In this case, it displays Loop #5.

The EVAL command takes the number in the ENV:Loop file as <value1>. <Value2> is 1, and the operation is subtraction. The output of the EVAL command is sent to the T:Qwe(\$\$) file. In this case, the value would be 4.

The next TYPE command sends the value in the T:Qwe(\$\$) file to the ENV:Loop file. In this case, it changes the value in ENV:Loop from 5 to 4.

The IF statement states that as long as the value for Loop is greater than 0, the script should start over. This results in the next line being Loop #4.

The script will continue until Loop is equal to 0.

EXCHANGE

Format: EXCHANGE [CX_POPKEY = <key>]
[CX_POPUP = no] [CX_PRIORITY = <n>]

Template: CX_POPKEY/K, CX_POPUP/K,
CX_PRIORITY/K/N

Purpose: To monitor and control the Commodity Exchange programs.

Path: SYS:Utilities/Exchange

Specification:

A list of acceptable key combinations can be found on page 5-29.

EXCHANGE is a Commodity Exchange program that monitors and controls all the other Commodity Exchange programs. CX_POPKEY = <key(s)> allows you to specify the hot key for the program. If more than one key is specified, be sure to enclose the entire argument in double-quotes (i.e., "CX_POPKEY = Shift F1").

CX_POPUP = no will keep the Exchange window from opening.

CX_PRIORITY = <n> sets the priority of Exchange in relation to all the other Commodity Exchange programs. All the Commodity Exchange programs are set to a default priority of 0.

To kill Exchange, press Ctrl-E.

Example:

```
1> EXCHANGE "CX_POPKEY = Shift F1"
```

The Exchange program will be started and its window will appear on the screen. If you Hide the window, then want to bring it back again, the hot key combination is Shift-F1.

EXECUTE

Format: EXECUTE <script> [{<arguments>}]

Template: (none)

Purpose: To execute a script with optional argument substitution.

Path: C:EXECUTE

Specification:

EXECUTE is used to run scripts of AmigaDOS commands. The lines in the script are executed just as if they had been typed at a Shell prompt. If the s protection bit of a file is set and the file is in the search path, you only need to type the filename—the EXECUTE command is not needed.

You can use parameter substitution in scripts by including special keywords in the script. When these keywords are used, you can pass variables to the script by including the variable in the EXECUTE command line. Before the script is executed, AmigaDOS checks the parameter names in the script against any arguments given on the command line. If any match, AmigaDOS substitutes the values you specified on the command line for the parameter name in the script. You can also specify default values for AmigaDOS to use if no variables are given. If you have not specified a variable, and there is no default specified in the script, then the value of the parameter is empty (no substitution is made).

The permissible keywords for parameter substitution are explained below. Each keyword must be prefaced with a dot character (.).

The .KEY (or .K) keyword specifies both keyword names and positions in a script. It tells EXECUTE how many parameters to expect and how to interpret them. In other words, .KEY serves as a template for the parameter values you specify. Only one .KEY statement is allowed per script. If present, it should be the first line in the file.

EXECUTE is generally made resident during the startup-sequence.

E

The arguments on the .KEY line can be given with the /A and /K directives, which work the same as in an AmigaDOS template. Arguments followed by /A are required; arguments followed by /K require the name of that argument as a keyword. For example, if a script starts with .KEY filename/A it indicates that a filename must be given on the EXECUTE command line after the name of the script. This filename will be substituted in subsequent lines of the script. For instance, if the first line of a script is:

```
.KEY filename/A, TOname/K
```

You must specify a filename variable. The TOname variable is optional, but if specified the TOname keyword must be used. For instance:

```
1> EXECUTE Script Textfile TOname NewFile
```

Before execution, AmigaDOS scans the script for any items enclosed by BRA and KET characters (< and >). Such items may consist of a keyword or a keyword and a default value. Wherever EXECUTE finds a keyword enclosed in angle brackets, it tries to substitute a parameter. However, if you want to use a string in your script file that contains angle brackets, you will have to define substitute "bracket" characters with the .BRA and .KET commands. .BRA <ch> changes the opening bracket character to <ch>, while .KEY changes the closing bracket character to <ch>. For example:

```
.KEY filename
ECHO "This line does NOT print <angle> brackets."
.BRA {
.KET }
ECHO "This line DOES print <angle> brackets."
ECHO "The specified filename is {filename}."
```

would result in the following output:

```
1> EXECUTE script TestFile
This line does NOT print brackets.
This line DOES print <angle> brackets.
The specified filename is TestFile.
```

The first ECHO statement causes AmigaDOS to look for a variable to substitute for the <angle> parameter. If no argument was given on the EXECUTE command line, the null string is substituted. The .BRA and .KET commands then tell the script to use braces to enclose parameters. So, when the second ECHO statement is executed, the angle brackets will be printed. The third ECHO statement illustrates that the braces now function as the bracket characters.

When enclosing a keyword in bracket characters, you can also specify a default string to be used if a variable is not supplied on the command line. There are two ways to specify a default. The first way requires that you specify the default every time you reference a parameter. You must separate the two strings with a dollar sign (\$).

For example, in the following statement:

```
ECHO "<word1$defword1> is the default for Word1."
```

defword1 is the default value specified for word1. It will be printed if no other variable is given for word1. However, if you want to specify this default several times in your script, you would have to use <word1\$defword1> each time.

The .DOLLAR <ch> command allows you to change the default character from \$ to <ch>. (You can also use .DOL <ch>.) For instance:

```
.DOL #  
ECHO "<word1#defword1> is the default for Word1."
```

The second way to define a default uses the .DEF command. This allows you to specify a default for each specific keyword.

For example:

```
.DEF word1 "defword1"
```

assigns defword1 as the default for the word1 parameter throughout the script. The following statement:

```
ECHO "<word1> is the default for Word1."
```

results in the same output as the previous ECHO statement:

```
defword1 is the default for Word1.
```

You can embed comments in a script by including them after a semicolon (;) or by typing a dot (.), followed by a space, then the comment.

Summary of Dot Commands	
.KEY	Argument template used to specify the format of arguments; may be abbreviated to .K
.DOT <ch>	Change dot character from . to <ch>
.BRA <ch>	Change opening "bracket" character from < to <ch>
.KET <ch>	Change closing "bracket" character from > to <ch>
.DOLLAR <ch>	Change default character from \$ to <ch>; may be abbreviated to .DOL
.DEF <keyword value>	Give default to parameter
.<space>	Comment line
.\	Blank comment line

When you EXECUTE a command line, AmigaDOS looks at the first line of the script. If it starts with a dot command, AmigaDOS scans the script looking for parameter substitution and builds a temporary file in the T: directory. If the file does not start with a dot command, AmigaDOS assumes that no parameter substitution is necessary and starts executing the file immediately without copying it to T:. If you do not need parameter substitution, do not use dot commands as they require extra disk accesses and increase execution time.

AmigaDOS provides a number of commands that are useful in scripts, such as IF, ELSE, SKIP, LAB, and QUIT. These commands, as well as the EXECUTE command, can be nested in a script. That is, a script can contain EXECUTE commands.

To stop the execution of a script, press Ctrl-D. If you have nested script files, you can stop the set of EXECUTE commands by pressing Ctrl-C. Ctrl-D only stops the current script from executing.

The current Shell number can be referenced by the characters <\$\$>. This is useful in creating unique temporary files, logical assignments, and PIPE names.

Examples:

Assume the script List contains the following:

```
.K filename
RUN COPY <filename> TO PRT: +
ECHO "Printing of <filename> done"
```

The following command

```
1> EXECUTE List Test/Prg
```

acts as though you had typed the following commands at the keyboard:

```
1> RUN COPY Test/Prg TO PRT: +
1> ECHO "Printing of Test/Prg done"
```

Another example, Display, uses more of the features described above:

```
.Key name/A
IF EXISTS <name>
TYPE <name> NUMBER ;if the file is in the given directory,
                    ;type it with line numbers

ELSE
ECHO "<name> is not in this directory"
ENDIF
```

The command:

```
1> RUN EXECUTE Display Work/Prg2
```

should display the Work/Prg2 file, with line numbers on the screen, if it exists on the current directory. If the file is not there, the screen displays an error message. Because of the /A, if a filename is not given on the command line after display, an error will occur.

See also: IF, SKIP, FAILAT, LAB, ECHO, RUN, QUIT

FAILAT

Format: FAILAT [<n>]

Template: RCLIM/N

Purpose: To instruct a command sequence to fail if a program gives a return code greater than or equal to the given value.

Path: Internal

Specification:

Commands indicate that they have failed in some way by setting a return code. A nonzero return code indicates that the command has encountered an error of some sort. The return code, normally 5, 10, or 20, indicates how serious the error

was. A return code greater than or equal to a certain limit, the *fail limit*, terminates a sequence of non-interactive commands (commands you specify after RUN or in a script).

You may use the FAILAT command to alter the fail limit RCLIM (Return Code Limit) from its initial value of 10. If you increase the limit, you indicate that certain classes of error should not be regarded as fatal and that execution of subsequent commands may proceed after an error. The argument must be a positive number. The fail limit is reset to the initial value of 10 on exit from the command sequence.

If the argument is omitted, the current fail limit is displayed.

Example:

Assume a script contains the following lines:

```
COPY DF0:MyFile to RAM:
ECHO "MyFile being copied."
```

If MyFile cannot be found, the script will be aborted and the following message will appear in the Shell window:

```
COPY: object not found
COPY failed returncode 20:
```

However, if you changed the return code limit to higher than 20, the script would continue even if the COPY command fails. For instance, if you changed the script to read:

```
FAILAT 21
COPY DF0:MyFile to RAM:
ECHO "MyFile being copied."
```

even if MyFile cannot be found, the script will continue. The following message will appear in the Shell window:

```
COPY: object not found
MyFile being copied.
```

See also: ECHO, EXECUTE

FAULT

Format: FAULT <error number(s)>

Template: /N,/N,/N,/N,/N,/N,/N,/N,/N,/N

Purpose: To print the message(s) for the specified error code(s).

Path: Internal

Specification:

A complete list of error messages starts on page 8-131.

FAULT prints the message(s) corresponding to the error number(s) supplied. Up to ten error numbers can be specified at once. If several error numbers are given with FAULT, they may be separated by commas or spaces.

Example:

If you received the error message Error when opening DF1:TestFile 205 and needed more information, you would type:

```
1> FAULT 205
FAULT 205: object not found
```

This tells you that the error occurred because TestFile could not be found on DF1:.

FILENOTE

Format: FILENOTE [FILE] <file|pattern> [[COMMENT] <comment>] [ALL] [QUIET]

Template: FILE/A,COMMENT,ALL/S,QUIET/S

Purpose: To attach a comment to a file.

Path: C:FILENOTE

Specification:

FILENOTE attaches an optional comment of up to 79 characters to the specified file or to all files matching the given pattern.

If the <comment> includes spaces, it must be enclosed in double quotes. To include double quotes in a filenote, each literal quote mark must be immediately preceded by an asterisk (*), and the entire comment must be enclosed in quotes, *regardless of whether the comment contains any spaces.*

If the <comment> argument is omitted, any existing filenote will be deleted from the named file.

Creating a comment with FILENOTE is the same as entering a comment into the Comment gadget of an icon's Information window. Changes made with FILENOTE will be reflected in the Information window, and vice versa.

When an existing file is copied to (specified as the TO argument of a COPY command), it will be overwritten, but its comment will be retained. Any comment attached to a FROM file will not be copied unless the CLONE or COM option of COPY is specified.

If the ALL option is given, FILENOTE will add the <comment> to all the files in the specified directory. If the QUIET option is given, screen output is suppressed.

Examples:

```
1> FILENOTE Sonata "allegro non troppo"
```

attaches the filenote *allegro non troppo* to the Sonata file.

```
1> FILENOTE Toccata ""*""presto*""
```

Here the filenote is *"presto"*.

FIXFONTS

Format: FIXFONTS

Template: (none)

Purpose: To update the .font files of the FONTS: directory.

Path: SYS:System/FixFonts

Specification:

FIXFONTS runs the FixFonts program. (FIXFONTS does not support any arguments.) Your disk light will come on while the FONTS: directory is updated. When the update is finished, the light will go out and a Shell prompt will appear.

Example:

```
1> FIXFONTS
```

FKEY

Format: FKEY [F1-F10=<string>] [SF1-SF10=<string>]
[CX_PRIORITY=n] [CX_POPUP=yes|no]
[CX_POPKEY=<key>]

Template: KEY,CX_PRIORITY/K/N,CX_POPUP/K,
CX_POPKEY/K

Purpose: To assign text to function and shifted functions keys.

Path: Extras2.0:Tools/Commodities/FKey

Specification:

FKEY is a Commodities Exchange program that allows you to assign a text string to the function keys and shifted function keys. The output of the function keys is viewable through the Execute Command menu item or in a Shell window.

Example:

```
1> RUN FKEY F4=INFO\n CX_POPUP=no
```

assigns the INFO command to the F4 key. The FKey program will be started but the CX_POPUP=no option keeps the window from opening. Pressing F4 while working in a Shell window is the same as typing the INFO command and pressing Return.

FONT

F

Format: FONT [FROM <filename>] [EDIT] [USE] [SAVE] [WORKBENCH] [SCREEN] [SYSTEM]

Template: FROM,EDIT/S,USE/S,SAVE/S,WORKBENCH/S,SCREEN/S,SYSTEM/S

Purpose: To specify the font(s) used by the system.

Path: SYS:Prefs/Font

Specifications:

FONT with no arguments or with the EDIT argument opens the Font editor.

The FROM argument must be used in combination with at least one WORKBENCH, SCREEN, or SYSTEM switch. (You can use more than one switch.) This allows you to specify a particular font to be used in the designated area(s) of the screen. The FROM file must be one that was previously saved with the Save As menu item of the Font editor's Project menu. Even if the font in the FROM file was originally saved as one type of text, it can be used in a different area of the screen by specifying the appropriate switch. For instance, if the FROM file was created when you saved a font as Screen text, that font can be used as the Workbench icon text by specifying the WORKBENCH switch after the filename.

If you specify the USE option, the font will be loaded into the appropriate area and used, just as if you had opened the Font editor, selected the appropriate radio button, chosen the font, and selected the Use gadget. If you specify the SAVE option, that font will be saved.

If you do not specify USE or SAVE, EDIT is assumed, and the Font editor is opened. If a FROM file and a WORKBENCH, SCREEN, or SYSTEM switch is specified, the Font editor will open with the font saved in the FROM file displayed next to the selected radio button(s). If no switch is specified with the FROM file, the editor will display the last used configuration.

Examples:

1> FONT Prefs/Presets/Font.screen WORKBENCH

opens the Font editor. The font previously saved in the Font.screen file will be displayed in the Workbench icon text gadget. You must select the Save, Use or Cancel gadget to close the editor.

1> FONT Prefs/Presets/Font.screen WORKBENCH USE

uses the font saved in the Font.screen file as the Workbench icon text. The Font editor is not opened. The font choice will be lost if the system is rebooted.

1> FONT Prefs/Presets/Font.screen SCREEN WORKBENCH USE

uses the font saved in the Font.screen file as both the Screen text and the Workbench icon text.

FORMAT

Format: FORMAT DRIVE <drive> NAME <name>
 [NOICONS] [QUICK] [FFS] [NOFFS]

Template: DRIVE/A/K,NAME/A/K,NOICON/S,QUICK/S,
 FFS/S,NOFFS/S

Purpose: To format a disk for use with the Amiga.

Path: SYS:System/Format

Specification:

To format a disk, you must specify both the DRIVE and the NAME keywords. The name can be from one to thirty-one characters in length. If you include spaces in the name, it must be enclosed in double quotes.

The NOICONS option prevents a Trashcan icon from being added to the newly formatted disk.

The QUICK option specifies that FORMAT will only format and create the root block (and track), the boot block (and track), and create the bitmap blocks. This is useful when reformatting a previously formatted floppy disk.

Normally, floppy disks are formatted with the old file system. For hard disks, FORMAT uses information specified by the HDToolbox program or in the MountList to determine the DOS type and file system. The FFS option marks the disk as being used with the FastFilesystem and overrides the MountList keywords or any other default file systems.

Examples:

```
1> FORMAT DRIVE DF0: NAME EmptyDisk
```

formats the disk in drive DF0:, erases any data, and names the disk EmptyDisk.

To reformat, or erase, a disk that already contains data, use the QUICK option.

```
1> FORMAT DRIVE DF2: NAME NewDisk QUICK
```

GET

Format: GET <name>

Template: NAME/A

Purpose: To get the value of a local variable.

Path: Internal

Specification:

GET is used to retrieve and display the value of a local environment variable. The value is displayed in the current window.

Local environment variables are only recognized by the Shell in which they are created, or by any Shells created from a NEWSHELL command executed in the original Shell. If you open an additional Shell by opening the Shell icon or by using the Execute Command menu item, previously created local environment variables will not be available.

Example:

```
1> GET editor
Extras2.0:Tools/MEmacs
```

See also: SET

GETENV

Format: GETENV <name>

Template: NAME/A

Purpose: To get the value of a global variable.

Path: Internal

Specification:

GETENV is used to retrieve and display the value of a global environment variable. The value is displayed in the current window. Global variables are stored in ENV: and are recognized by all Shells.

Example:

```
1> GETENV editor
Extras2.0:Tools/MEmacs
```

See also: SETENV

GRAPHICDUMP

Format: GRAPHICDUMP [TINY|SMALL|MEDIUM|
LARGE|<xdots>:<ydots>]

Template: TINY/S,SMALL/S,MEDIUM/S,LARGE/S,
<xdots>:<ydots>/S

Purpose: To print the frontmost screen.

Path: Extras2.0:Tools/GraphicDump

Specification:

GRAPHICDUMP sends a dump of the frontmost screen to the printer about ten seconds after issuing the command. The size options, which correspond to the program's acceptable Tool Types, determine the width of the printout:

TINY	1/4 the total width allowed by the printer
SMALL	1/2 the total width allowed by the printer
MEDIUM	3/4 the total width allowed by the printer
LARGE	the full width allowed by the printer

The height of the printout is such that the perspective of the screen is maintained.

To specify specific dimensions, substitute the absolute width in dots for <xdots> and the absolute height for <ydots>.

Examples:

```
1> GRAPHICDUMP SMALL
```

will produce a printout of the frontmost screen that is about one-half the total width allowed by the printer.

```
1> GRAPHICDUMP 600:300
```

will produce a printout that is 600 dots wide by 300 dots high.

ICONEDIT

Format: ICONEDIT

Template: (none)

Purpose: To edit the appearance and type of icons.

Path: Extras2.0:Tools/IconEdit

Specification:

ICONEDIT opens the IconEdit program. The command does not support any arguments.

Example:

```
1> ICONEDIT
```

ICONTROL

Format: ICONTROL [FROM <filename>] [EDIT] [USE] [SAVE]

Template: FROM,EDIT/S,USE/S,SAVE/S

Purpose: To specify parameters used by the Workbench.

Path: SYS:Prefs/IControl

Specification:

ICONTROL without any arguments or with the EDIT argument opens the IControl editor. The FROM argument lets you specify a file to open. This must be a file that was previously saved with the Save As menu item of the IControl editor. For instance, if you have saved a special configuration of the IControl editor to a file in the Presets drawer, you can use the FROM argument to open that file. If the USE switch is also given, the editor will not open, but the settings in the FROM file will be used. If the SAVE switch is given, the editor will not open, but the settings in the FROM file will be saved.

Example:

```
1> ICONTROL Prefs/Presets/IconControl.pre USE
```

uses the settings that were saved in the IControl.pre file. The editor is not opened.

ICONX

Format: ICONX

Template: (none)

Purpose: To allow execution of a script file from an icon.

Path: C:ICONX

Specification:

ICONX allows you to execute a script file of AmigaDOS commands via an icon.

To use ICONX, create or copy a project icon for the script. Open the icon's Information window and change the Default Tool of the icon to C:ICONX. Add the WINDOW = and DELAY = Tool Types if you choose, and select Save to store the changed .info file. The script can then be executed by double-clicking on the icon.

When the icon is opened, ICONX changes the current directory to the directory containing the project icon before executing the script. An input/output window for the script file will be opened on the Workbench screen. The icon's WINDOW = Tool Type can be used to specify the size of the window. You can add a delay (specified in seconds) after the execution of the file is complete with the DELAY = Tool Type. This will keep the window open to allow time for reading the output. If a 0 is specified for DELAY =, ICONX waits for a Ctrl-C before exiting.

Complete information on the WINDOW = specifications is given on page 7-38.

Extended selection can be used to pass files that have icons to the script. Their filenames appear to the script as keywords. To use this facility, the .KEY keyword must appear at the start of the script. In this case, the AmigaDOS EXECUTE command is used to execute the script file.

See Also: EXECUTE

IF

Format: IF [NOT] [WARN] [ERROR] [FAIL] [<string>
EQ|GT|GE <string>] [VAL] [EXISTS
<filename>]

Template: NOT/S,WARN/S,ERROR/S,FAIL/S,,EQ/K,
GT/K,GE/K,VAL/S,EXISTS/K

Purpose: To evaluate conditional operations in script files.

Path: Internal

Specification:

In a script file, IF, when its conditional is true, carries out all the subsequent commands until an ENDIF or ELSE command is found. When the conditional is not true, execution skips directly to the ENDIF or to an ELSE. The conditions and commands in IF and ELSE blocks can span more than one line before their corresponding ENDIFs.

Following are some of the ways you can use the IF, ELSE, and ENDIF commands:

IF <condition>	IF <condition>	IF <condition>
<command(s)>	<command(s)>	<command(s)>
ENDIF	ELSE	IF <condition>
	<command(s)>	<command(s)>
	ENDIF	ENDIF
		ENDIF

ELSE is optional, and nested IFs jump to the nearest ENDIF.

The additional keywords are as follows:

NOT	Reverses the interpretation of the result.
WARN	True if previous return code is greater than or equal to 5.
ERROR	True if previous return code is greater than or equal to 10; only available if you set FAILAT to greater than 10.
FAIL	True if previous return code is greater than or equal to 20; only available if you set FAILAT to greater than 20.
<a> EQ 	True if the text of a and b is identical (disregarding case).
EXISTS <file>	True if the file exists.

If more than one of the three condition-flag keywords (WARN, ERROR, FAIL) are given, the one with the lowest value is used.

IF supports the GT (greater than) and GE (greater than or equal to) comparisons. Normally, the comparisons are performed as string comparisons. However, if the VAL option is specified, the comparison is a numeric comparison.

NOTE: You can use NOT GE for LT and NOT GT for LE.

You can use local or global variables with IF by prefacing the variable name with a \$ character.

Examples:

```
IF EXISTS Work/Prog
TYPE Work/Prog
ELSE
ECHO "It's not here"
ENDIF
```

If the file Work/Prog exists in the current directory, then AmigaDOS displays it. Otherwise, AmigaDOS displays the message It's not here and continues after the ENDIF.


```
IF ERROR
SKIP errlab
ENDIF
ECHO "No error"
LAB errlab
```

If the previous command produced a return code greater than or equal to 10 then AmigaDOS skips over the ECHO command to the errlab label.

See also: EXECUTE, FAILAT, LAB, QUIT, SKIP

IHELP

Format: IHELP [CYCLE = <key>] [MAKEBIG = <key>]
[MAKESMALL = <key>] [CYCLEScreens =
<key>] [ZIPWINDOW = <key>]
[CX_PRIORITY = <n>]

Template: CYCLE/K, MAKEBIG/K, MAKESMALL/K,
CYCLEScreens/K, ZIPWINDOW/K,
CX_PRIORITY/K/N

Purpose: To enable the keyboard to take over certain mouse operations.

Path: Extras2.0:Tools/Commodities/IHelp

Specification:

A list of acceptable key combinations can be found on page 5-29.

IHELP is a Commodities Exchange program that lets you assign functions normally performed by the window gadgets to keys. The arguments supported by IHelp are the same as the Tool Types that can be entered into the icon's Information window. If a <key> argument specifies multiple keys, be sure to enclose the entire argument in double quotes. A list of the arguments follows:

CYCLE	Cycles any open tool or project screens from the back of the screen to the front.
MAKEBIG	Makes the active window as large as possible without moving it.

MAKESMALL	Makes the active window as small as possible.
CYCLEScreens	Cycles through all open screens.
ZIPWINDOW	Zooms the active window. (This is the same as selecting the window's zoom gadget.)

The `CX_PRIORITY = <n>` argument sets the priority of IHelp in relation to all the other Commodity Exchange programs. All the Commodity Exchange programs are set to a default priority of 0. For instance, if IHelp has a priority of 3, it will intercept any keys specified for the arguments before any other Exchange programs.

Example:

```
1>IHELP "CYCLE = Alt F7" "MAKESMALL = Control S" "MAKEBIG = Control B"
```

If you were to press Alt-F7, any project or tool windows would cycle from front to back. The Ctrl-S combination makes the selected window small; while the Ctrl-B combination makes the selected window bigger.

INFO

Format: INFO [<device>]

Template: DEVICE

Purpose: To give information about the file system(s).

Path: C:INFO

Specification:

INFO displays a line of information about each disk or partition. This includes the maximum size of the disk, the used and free space, the number of soft disk errors that have occurred, and the status of the disk.

With the DEVICE argument, INFO provides information on just one device or volume.

Example:

```
1> INFO
Unit  Size  Used  Free  Full  Errs  Status  Name
DF0: 879K 1738   20  98%   0    Read Only  Workbench2.0
DF1: 879K  418 1140  24%   0    Read/Write  Text-6
```

Volumes available:
Workbench2.0 [Mounted]
Text-6 [Mounted]

INITPRINTER

Format: INITPRINTER

Template: (none)

Purpose: To initialize a printer for print options specified in the Preferences editors.

Path: Extras2.0:Tools/InitPrinter

Specification:

INITPRINTER runs the InitPrinter program. (It does not support any arguments.) You will hear the printer reset, then the Shell prompt will return.

Example:

```
1> INITPRINTER
```

INPUT

Format: INPUT [FROM <filename>] [EDIT] [USE] [SAVE]

Template: FROM,EDIT/S,USE/S,SAVE/S

Purpose: To specify different speeds for the mouse and keyboard.

Path: SYS:Prefs/Input

Specification:

INPUT without any arguments or with the EDIT argument opens the Input editor. The FROM argument lets you specify a file to open. This must be a file that was previously saved with the Save As menu item of the Input editor. For instance, if you have saved a special configuration of the Input editor to a file in the Presets drawer, you can use the FROM argument to open that file. If the USE switch is also given, the editor will not be opened, but the settings in the FROM file will be used. If the SAVE switch is given, the editor will not open, but the settings in the FROM file will be saved.

Example:

```
1> INPUT Prefs/Presets/Input.fast SAVE
```

loads and saves the settings from the Input.fast file. Even if the system is rebooted, those settings will still be in effect. The editor does not open.

INSTALL

Format: INSTALL [DRIVE] <DF0:|DF1:|DF2:|DF3:>
[NOBOOT] [CHECK] [FFS]

Template: DRIVE/A,NOBOOT/S,CHECK/S,FFS/S

Purpose: To write the boot block to a formatted floppy disk, specifying whether it should be bootable.

Path: C:INSTALL

Specification:

INSTALL clears a floppy disk's boot block area and writes a valid boot block onto the disk. By default, the disk will be given the boot block of the filing system specified when the disk was initially formatted, either the old filing system (OFS) or the FastFileSystem (FFS). To force FastFileSystem, use the FFS switch.

The NOBOOT option removes the boot block from an AmigaDOS disk, making it not bootable.



The NOBOOT option will write a boot block on a non-AmigaDOS disk. INSTALL will use the default DOS type, OFS, when writing to a non-AmigaDOS disk.

The CHECK option checks for valid boot code. It reports whether a disk is bootable or not and whether standard Commodore-Amiga boot code is present on the disk. The condition flag is set to 0 if the boot code is standard (or the disk isn't bootable), 5 (WARN) otherwise.

Examples:

```
1> INSTALL DF0: CHECK
No bootblock installed
```

indicates that there is a non-bootable floppy in DF0:.

```
1> INSTALL DF0:
```

makes the disk in drive DF0: a bootable disk.

1> INSTALL DF0: CHECK
Appears to be FFS bootblock

indicates that there is an FFS floppy in DF0:.

IPREFS

Format: IPREFS

Template: (none)

Purpose: To communicate Preferences information stored in the individual editor files to the Workbench.

Path: C:IPREFS

Specifications:

IPREFS reads the individual system Preferences files and passes the information to the Workbench so that it can reply accordingly. IPREFS is generally run in the Startup-sequence after the Preferences files are copied to ENV:. Each time a user selects Save or Use from within an editor, IPREFS is notified and passes the information along to Workbench. If necessary, IPREFS will reset Workbench in order to implement those changes. If any project or tool windows are open, IPREFS will display a requester asking you to close any non-drawer windows.

JOIN

Format: JOIN {<file|pattern>} AS|TO <filename>

Template: FILE/M,AS = TO/K/A

Purpose: To concatenate two or more files into a new file.

Path: C:JOIN

Specification:

JOIN copies all the listed files, in the order given, to one new file. This destination file cannot have the same name as any of the source files. You must supply a destination filename. The

original files remain unchanged. Any number of files may be JOINed in one operation.

TO can be used as a synonym for AS.

Example:

```
1> JOIN Part1 Part2 Part3 AS Textfile
```

KEYSHOW

Format: KEYSHOW

Template: (none)

Purpose: To display the current Keymap.

Path: Extras2.0:Tools/KeyShow

Specification:

KEYSHOW opens the KeyShow window. (The command does not support any arguments.) To exit the program, select the window's close gadget.

Example:

```
1> KEYSHOW
```

LAB

Format: LAB [<string>]

Template: (none)

Purpose: To specify a label in a script file.

Path: Internal

Specification:

LAB is used in scripts to define a label that is looked for by the SKIP command. The label <string> may be of any length but must be alphanumeric characters. No symbols are allowed. If the <string> contains spaces, it must be enclosed in quotes.

See also: SKIP, IF, EXECUTE

LIST

Format: LIST [{<dir|pattern>}] [P|PAT <pattern>] [KEYS]
[DATES] [NODATES] [TO <name>] [SUB
<string>] [SINCE <date>] [UPTO <date>]
[QUICK] [BLOCK] [NOHEAD] [FILES] [DIRS]
[LFORMAT <string>] [ALL]

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

Purpose: To list specified information about directories and files.

Path: C:LIST

Specification:

LIST displays information about the contents of the current directory. If you specify a <dir>, <pattern>, or <filename> argument, LIST will display information about the specified directory, all directories or files that match the pattern, or the specified file, respectively.

Unless other options are specified, LIST displays the following:

name	The name of the file or directory.
size	The size of the file in bytes. If there is nothing in this file, the field will read empty. For directories, this entry reads Dir.
protection	The protection bits that are set for this file are shown as letters. The clear (unset) bits are shown as hyphens. Most files will show the default protection bits, ----rwd for readable/writable/executable/deleteable. See the PROTECT command for more on protection bits.

K**L**

date and time	The date and time the file was created or last altered.
comment	The comment, if any, placed on the file using the FILENOTE command. It is preceded by a colon (:).

LIST has options which will change the way the output is displayed. These options are explained below:

KEYS	Displays the block number of each file header or directory.
DATES	Displays dates in the form DD-MMM-YY (the default unless you use QUICK).
NODATES	Will not display date and time information.
TO <name>	Specifies an output file or device for LIST; by default, LIST outputs to the current window.
SUB <string>	Lists only files containing the substring <string>.
SINCE <date>	Lists only files created on or after a certain date.
UPTO <date>	Lists only files created on or before a certain date.
QUICK	Lists only the names of files and directories.
BLOCK	Displays file sizes in blocks, rather than bytes.
NOHEAD	Suppresses the printing of the header information.
FILES	Lists files only (no directories).
DIRS	Lists directories only (no files).

LFORMAT	Defines a string to specially format LIST output.
ALL	Lists all files in directories and subdirectories.

The LFORMAT option modifies the output of LIST and can be used as a quick method of generating script files. When LFORMAT is specified, the QUICK and NOHEAD options are automatically selected. When using LFORMAT you must specify an output format specification string; this string is incorporated into the resulting output. If you want the output to be saved, you must redirect it to a file by using the > operator or specifying a TO file.

The format for the output format specification string is LFORMAT = <string>. To include the output of LIST in this string, use the substitution operator %S. The path and filename can be made part of the string this way. Whether the path or the filename is substituted for an occurrence of %S depends on how many occurrences are in the LFORMAT line, and their order, as follows:

Occurrences of %S	Substituted with each occurrence			
	1st	2nd	3rd	4th
1	filename			
2	path	filename		
3	path	filename	filename	
4	path	filename	path	filename

When using %S, the path is always relative to the current directory.

Some new operators allow you to specify fields to be printed in the LFORMAT output. These operators are:

%A	Prints file attributes (protection bits).
%B	Prints size of file in blocks.
%C	Prints any comments attached to the file.
%D	Prints the date associated with the file.
%F	Prints the complete file parent path, regardless of the current directory.
%K	Prints the file key block .

%L	Prints the length of file in bytes.
%N	Prints the name of the file.
%P	Prints the file parent path relative to the current directory.
%T	Prints the time associated with the file.

You can put a length specifier and/or a justification specifier between the percent sign (%) and the field specifier.

Examples:

```
1> LIST Dirs
Monitors   Dir ---- rwed 27-June-90 11:43:59
T          Dir ---- rwed Wednesday 11:37:43
Trashcan   Dir ---- rwed 21-Jun-90 17:54:20
```

Only the directories in the current directory, in this case SYS:, are listed. (A shortened version of the output is shown above.)

```
1> LIST Li#? TO RAM:Libs.file
```

LIST will search for any directories or files that start with LI. The output of LIST will be sent to the Libs.file in RAM:.

```
1> LIST DF0:Documents UPTO 09-Oct-90
```

Only the files or directories on the Documents directory of DF0: that have not been changed since October 9, 1990, will be listed.

```
1> LIST >RAM:Scriptnotes #? LFORMAT = "filenote %S%S Testnote"
```

A new script file, Scriptnotes, is created in RAM:. The contents will include a list of all the files in the current directory. When Scriptnotes is executed, it will add the filenote Testnote to each file.

```
1> LIST TestFile LFORMAT "%-25N %6L %A %-9D %T*N: %C"
TestFile      28 ----rwed 28-May-91 14:45:40
:
```

The output is organized in the same way as the default LIST output.

LOADWB

Format: LOADWB [-DEBUG] [DELAY] [CLEANUP]
[NEWPATH]

Template: -DEBUG/S, DELAY/S, CLEANUP/S,
NEWPATH/S

Purpose: To start Workbench.

Path: C:LOADWB

Specification:

LOADWB starts the Workbench. Normally, this is done when booting, by placing the LOADWB command in the Startup-sequence file. If you shut down the Workbench, LOADWB can be used from a Shell to restart it.

The -DEBUG option makes a special developer menu, Debug, available in the Workbench menu bar. If the DELAY option is specified, LOADWB waits three seconds before executing, giving disk activity time to stop. The CLEANUP option automatically performs a "cleanup" of the window.

Workbench snapshots the current paths in effect when the LOADWB command is executed. It uses these paths for each Shell started from Workbench. NEWPATH allows you to specify a new path which is snapshot from the current Shell.

Example:

If you have quit the Workbench and are working through a Shell, typing:

```
1> LOADWB
```

will bring the Workbench back. Typing LOADWB when the Workbench is already loaded has no effect.

```
1> PATH DF2:Bin ADD
```

```
1> LOADWB NEWPATH
```

loads Workbench. Any Shells started from the icon will have the same path as the Shell used to run the LOADWB NEWPATH command.

LOCK

Format: LOCK <drive> [ON|OFF] [<passkey>]

Template: DRIVE/A,ON/S,OFF/S,PASSKEY

Purpose: To set the write protect status of a disk.

Path: C:LOCK

Specification:

LOCK sets or unsets the write protect status of a disk or partition. The LOCK remains on until the system is rebooted or until the LOCK is turned off with the LOCK OFF command.

An optional passkey may be specified. If the passkey is used to lock a hard disk partition, the same passkey must be specified to unlock the partition. The passkey may be any number of characters in length.

Example:

```
1> LOCK Work: ON SecretCode
```

The Work: partition is locked. You can read the contents of Work: with commands like DIR, LIST or MORE, but you cannot alter the contents of the partition. If you try to edit the contents of a file on Work:, a requester will appear stating that Work: is write-protected. For instance, if you try to create a new directory, the following message will appear:

```
1> MAKEDIR WORK:Test
Can't create directory Work:Test
Disk is write-protected
```

To unlock the partition, type:

```
1> LOCK Work: OFF SecretCode
```

MAGTAPE

Format: MAGTAPE [DEVICE <devicename>]
[UNIT <n>] [RET|RETENSION]
[REW|REWIND] [SKIP <n>]

Template: DEVICE/K,UNIT/N/K,RET = RETENSION/S,
REW = REWIND/S,SKIP/N/K

Purpose: To retension, rewind, or skip forward on SCSI tapes.

Path: C:MAGTAPE

Specification:

By default, MAGTAPE uses scsi.device unit 4. To change the default, you must use both the DEVICE and UNIT keywords.

The RET|RETENSION option runs the tape to the end, then rewinds it. The REW|REWIND options rewind the tape. The SKIP <n> option allows you to skip files on the tape.

MAGTAPE tests to see if the unit is ready before sending the command. If your tape is not on-line, you may have to repeat the MAGTAPE command.

Example:

```
1> MAGTAPE DEVICE second_scsi.device UNIT 0 REW
```

MAKEDIR

Format: MAKEDIR {<name>}

Template: NAME/M

Purpose: To create a new directory.

Path: C:MAKEDIR

Specification:

MAKEDIR creates a new, empty directory(s) with the name(s) you specify. The command works within only one directory level at a time, so any directories on the given path(s) must already exist. The command fails if a directory or a file of the same name already exists in the directory above it in the hierarchy. MAKEDIR does not create a drawer icon for the new directory.

Examples:

1> MAKEDIR Tests

creates a directory Tests in the current directory.

1> MAKEDIR DF1:Xyz

creates a directory Xyz in the root directory of the disk in DF1:.

1> CD DF0:

1> MAKEDIR Documents Payables Orders

creates three directories, Documents, Payables, and Orders, on the disk in DF0:.

MAKELINK

Format: MAKELINK [FROM] <file> [TO] <file> [HARD]
[FORCE]

Template: FROM/A,TO/A,HARD/S,FORCE/S

Purpose: To create a link between files.

Path: C:MAKELINK

Specification:

MAKELINK creates a file on a disk that is a pointer to another file, this is known as a **link**. When an application or command calls the FROM file, the TO file is actually used. By default, MAKELINK supports hard links—the FROM file and TO file must be on the same volume.

NOTE: Soft links (symbolic links), which can be links across volumes, are not currently supported.

Normally, MAKELINK does not support directory links, as they can be dangerous to applications. To create a directory link, you must use the FORCE option. If MAKELINK detects that you are creating a circular link, such as a link to a parent directory, you will receive a Link loop not allowed message.

MORE

Format: MORE <filename>

Template: FILENAME/K

Purpose: To display the contents of an ASCII file.

Path: SYS:Utilities/More

Specification:

MORE displays the contents of the file <filename>. If the file is not in the current directory, you must specify the complete path. If you don't specify a file, MORE will display a file requester.

MORE also accepts input from a PIPE. Since standard input from the Pipe-Handler is of unknown length, the Backspace, >, and %n commands are disabled when the MORE input is from a PIPE.

If the EDITOR environment variable is defined and you are using MORE from the Shell, you can bring up an editor to use on the file you are viewing (type Shift-E). The EDITOR variable should have the complete path to the editor specified, i.e. C:ED.

Example:

```
1> MORE DF0:TestFile
```

displays the contents of the ASCII file called TestFile on the disk in drive DF0:.

MOUNT

Format: MOUNT <device> [FROM <filename>]

Template: DEVICE/A, FROM/K

Purpose: To make a device connected to the system available.

Path: C:MOUNT

Specification:

MOUNT causes AmigaDOS to recognize devices connected to the system. When the MOUNT command is issued, MOUNT looks in the DEVS:MountList file (or the optional FROM file) for the parameters of the device that is being mounted.

MOUNT commands are usually placed in the Startup-sequence file.

Example:

Sample uses of MOUNT in the startup-sequence, include:

MOUNT Speak:

MOUNT Aux:

MOUNT Pipe:

These commands MOUNT the Speak.handler, Aux.handler, and Pipe.handler found in the L: directory.

See also: Chapter 7, "Using AmigaDOS."

NEWCLI

Format: NEWCLI [<window specification>]
[FROM <filename>]

Template: WINDOW, FROM

Purpose: To start a new Shell process.

Path: Internal

Specifications:

NEWCLI starts a new Shell process. It is the same as using the NEWSHELL command. See the specifications for NEWSHELL for more information.

NEWSHELL

Format: NEWSHELL [<window specification>]
[FROM <filename>]

Template: WINDOW, FROM

Purpose: To open a new interactive Shell window.

Path: Internal

Specifications:

NEWSHELL invokes a new, interactive Shell. The new window becomes the currently-selected window and process. The new window has the same current directory, prompt string, and stack size as the one from which it was invoked. However, each Shell window is independent, allowing separate input, output, and program execution.

The window can be sized, dragged, zoomed, and depth-adjusted just like most other Amiga windows.

To create a custom window, you can include the WINDOW argument. You may specify the initial dimensions, location, and title of the window with this <window specification> syntax:

CON:x/y/width/height/title/options

where:

- | | |
|--------|--|
| x | Is the number of pixels from the left edge of the screen to the left border of the Shell window. |
| y | Is the number of pixels from the top of the screen to the top of the Shell window. |
| width | Is the width of the Shell window, in pixels. |
| height | Is the height of the Shell window, in pixels. |
| title | Is the text that appears in the Shell window title bar. |

For a full explanation of the available options, see the "Customizing the Window" section of Chapter 7 (page 7-38).

NEWSHELL uses the default startup file S:Shell-startup, unless a FROM filename is specified. You might have several different Shell-startup files, each having different command aliases, for example. You can call such customized Shell environments with FROM.

Examples:

1> NEWSHELL

a new Shell window will open.


```
1> NEWSHELL CON:0/0/640/200/MyShell/CLOSE
```

a window starting in the upper left corner of the screen and measuring 640 pixels wide and 200 pixels high will open. The window will be titled MyShell, and it will have a close gadget. If you add the command to your User-startup file, a Shell window will open automatically when your Amiga is booted.

```
1> NEWSHELL FROM S:Programming.startup
```

opens a new Shell, but instead of executing the Shell-startup file, the Programming.startup file is executed. You could have aliases and prompt commands in the Programming-startup file that you only use when you are programming.

NOCAPSLOCK

Format: NOCAPSLOCK [CX_PRIORITY = <n>]

Template: CX_PRIORITY/K/N

Purpose: To disable the Caps Lock key.

Path: Extras2.0:Tools/Commodities/NoCapsLock

NOCAPSLOCK is a Commodity Exchange program that temporarily disables the Caps Lock key.

CX_PRIORITY = <n> sets the priority of NoCapsLock in relation to all the other Commodity Exchange programs. All the Commodity Exchange programs are set to a default priority of 0.

To kill NoCapsLock, press Ctrl-E.

Example:

```
1> NOCAPSLOCK
```

NOFASTMEM

Format: NOFASTMEM

Template: (none)

Purpose: To force the Amiga to use only resident Chip RAM.

Path: SYS:System/NoFastMem

Specification:

NOFASTMEM disables any Fast (or expansion) RAM used by the system. The expansion memory can be turned on again by sending the NoFastMem program a break, either via the BREAK command or by typing Ctrl-C. Ctrl-C will only work if you don't start the program with the RUN command.

Example:

```
1> NOFASTMEM
```

OVERSCAN

Format: OVERSCAN [FROM <filename>] [EDIT] [USE]
[SAVE]

Template: FROM,EDIT/S,USE/S,SAVE/S

Purpose: To change the sizes of the display areas for text and graphics.

Path: SYS:Prefs/Overscan

Specification:

OVERSCAN without any arguments or with the EDIT argument opens the Overscan editor. The FROM argument lets you specify a file to open. This must be a file that was previously saved with the Save As menu item of the Overscan editor. For instance, if you have saved a special configuration of the Overscan editor to a file in the Presets drawer, you can use the FROM argument to open that file. If the USE switch is also

N**O**

given, the editor will not open, but the settings in the FROM file will be used. If the SAVE switch is given, the editor will not open, but the settings in the FROM file will be saved.

Example:

```
1> OVERSCAN Prefs/Presets/Overscan.graphics SAVE
```

loads and saves the Overscan sizes saved in the Overscan.graphics file.

PALETTE

Format: PALETTE [FROM <filename>] [EDIT] [USE]
[SAVE]

Template: FROM,EDIT/S,USE/S,SAVE/S

Purpose: To change the colors of the Workbench screen.

Path: SYS:Prefs/Palette

Specification:

PALETTE without any arguments or with the EDIT argument opens the Palette editor. The FROM argument lets you specify a file to open. This must be a file that was previously saved with the Save As menu item of the Palette editor. For instance, if you have saved a special configuration of the Palette editor to a file in the Presets drawer, you can use the FROM argument to open that file. If the USE switch is also given, the editor will not be opened, but the settings in the FROM file will be used. If the SAVE switch is given, the editor will not open, but the settings in the FROM file will be saved.

Example:

```
1> PALETTE Prefs/Presets/Palette.grey USE
```

loads and uses the colors saved in the Palette.grey file. If the system is rebooted, the previously saved colors will be used.

PATH

Format: PATH [{<dir>}] [ADD] [SHOW] [RESET] [QUIET]
[REMOVE]

Template: DIR/M,ADD/S,SHOW/S,RESET/S,QUIET/S,
REMOVE/S

Purpose: To control the directory list that the Shell
searches to find commands.

Path: Internal

Specification:

PATH lets you see, add to, or change the search path that AmigaDOS follows when looking for a command or program to execute. When a directory is in the search path, you no longer need to specify the complete path to any files or subdirectories within that directory. You can just enter the filename, and AmigaDOS will look through the directories in the search path until it finds the file.

Enter the PATH command alone, or with the SHOW option, and the directory names in the current search path will be displayed. Normally, when PATH is displaying the directory names, a requester will appear if a volume that is part of the search path cannot be found. For instance, if you added a floppy disk to the search path, then removed that disk from the disk drive, a requester would ask you to insert the disk.

If you specify the QUIET option, PATH will not display requesters for volumes that are not currently mounted. If PATH encounters an unmounted volume, it will simply display the volume name. The names of any directories on that volume included in the PATH will not be displayed.

The ADD option specifies directory names to be added to the current PATH. You can add up to ten directories with one PATH ADD command (the ADD keyword is optional); names of the directories must be separated by at least one space.

When you issue the PATH command, AmigaDOS searches for each of the ADDED directories.

To replace the existing search path with a completely new one, use PATH RESET followed by the names of the directories. The existing search path, except for the current directory and SYS:C, is erased and the new one is substituted.

The REMOVE option eliminates the named directory from the search path.

Examples:

```
1> PATH EXTRAS2.0:Tools ADD
```

add the Tools directory on the Extras2.0 disk to the search path of the Shell. If the Extras2.0 disk is not in a disk drive, a requester will ask you to insert it in any drive.

If you remove Extras2.0 from the drive, and type:

```
1> PATH
```

a list of directories in the search path will be displayed. A requester will ask you to insert Extras2.0. However, if you had typed:

```
1> PATH QUIET
```

the list of directories in the search path will be displayed; however when the path comes to Extras2.0:Tools, only the volume name, Extras2.0:, will appear in the list.

See also: ASSIGN

POINTER

Format: POINTER [FROM <filename>] [EDIT] [USE]
[SAVE]

Template: FROM,EDIT/S,USE/S,SAVE/S

Purpose: To change the appearance of the screen pointer.

Path: SYS:Prefs/Pointer

Specification:

POINTER without any arguments or with the EDIT argument opens the Pointer editor. The FROM argument lets you specify a file to open. This must be a file that was previously saved with the Save As menu item of the Pointer editor. For instance, if you have saved a special configuration of the Pointer editor to a file in the Presets drawer, you can use the FROM argument to open that file. If the USE switch is also given, the editor will not be opened, but the settings in the FROM file will be used. If the SAVE switch is given, the editor will not open, but the settings in the FROM file will be saved.

Example:

```
1> POINTER Prefs/Presets/Pointer.star USE
```

loads and uses the pointer saved in the Pointer.star file. If the system is rebooted, the previously saved pointer will appear.

PRINTER

Format: PRINTER [FROM <filename>] [EDIT] [USE]
[SAVE]

Template: FROM,EDIT/S,USE/S,SAVE/S

Purpose: To specify a printer and print options.

Path: SYS:Prefs/Printer

Specification:

PRINTER without any arguments or with the EDIT argument opens the Printer editor. The FROM argument lets you specify a file to open. This must be a file that was previously saved with the Save As menu item of the Printer editor. For instance, if you have saved a special configuration of the Printer editor to a file in the Presets drawer, you can use the FROM argument to open that file. If the USE switch is also given, the editor will not be opened, but the settings in the FROM file will be used. If the SAVE switch is given, the editor will not open, but the settings in the FROM file will be saved.

Example:

```
1> PRINTER Prefs/Presets/Printer.epson SAVE
```

loads and saves the specifications saved in the Printer.epson file.

PRINTERGFX

Format: PRINTERGFX [FROM <filename>] [EDIT] [USE]
[SAVE]

Template: FROM,EDIT/S,USE/S,SAVE/S

Purpose: To specify graphic printing options.

Path: SYS:Prefs/PrinterGfx

Specification:

PRINTERGFX without any arguments or with the EDIT argument opens the PrinterGfx editor. The FROM argument lets you specify a file to open. This must be a file that was

previously saved with the Save As menu item of the PrinterGfx editor. For instance, if you have saved a special configuration of the PrinterGfx editor to a file in the Presets drawer, you can use the FROM argument to open that file. If the USE switch is also given, the editor will not be opened, but the settings in the FROM file will be used. If the SAVE switch is given, the editor will not open, but the settings in the FROM file will be saved.

Example:

```
1> PRINTERGFX Prefs/Presets/PrinterGfx.halftone USE
```

loads and uses the specifications saved in the PrinterGfx.halftone file. If the system is rebooted, the last saved specifications will be loaded.

PRINTFILES

Format: PRINTFILES {[-f] <filename>}

Template: -f/S,FILENAME/A/M

Purpose: To send file(s) to the printer.

Path: Extras2.0:Tools/PrintFiles

Specification:

PRINTFILES prints the specified file. The -f flag turns on the form feed mode. When printing multiple files, be sure to specify the flag before each filename.

Example:

```
1> PRINTFILES -f DF0:testfile -f DF0:docfile
```

prints the testfile and docfile files, stored on the disk inserted in drive DF0:. The -f argument will add a form feed between the two files so that they each start on a new page.

PROMPT

Format: PROMPT [<prompt>]

Template: PROMPT

Purpose: To change the prompt string of the current Shell.

Path: Internal

Specification:

In the examples in this manual, the prompt string is shown as 1>.

PROMPT allows you to customize the prompt string, the text printed by the Shell at the beginning of a command line. The prompt string may contain any characters, including escape sequences.

The default prompt string is:

`"%N.%S> "` Displays the Shell number, a period, the current directory, a right angle-bracket *and* a space.

The substitutions available for the <prompt> string are:

%N	Displays the Shell number.
%S	Displays the current directory.
%R	Displays the return code for the last operation.

A space is not automatically added to the end of the string. If you want a space between the prompt and typed-in text, place it in the string, and enclose the string in double quotes.

You can embed commands in the prompt string by enclosing the command in backward quotes (`).

PROMPT alone, without a string argument, resets the prompt to the default.

Examples:

```
1> PROMPT %N
1
```

Only the Shell number is shown. The > is removed from the prompt.

```
1> PROMPT "%N.%S.%R> "
1.SYS:.0>
```

The Shell number, current directory, and return code of the previous command are shown. A space is included after the >.

```
1> PROMPT "'date'>"
Tuesday 11-Sep-90 14:36:39>
```

The DATE command is executed and used as the prompt. The prompt is not updated as the time changes. You would have to execute the PROMPT command again to update the Shell prompt.

PROTECT

Format: PROTECT [FILE] <file|pattern> [FLAGS]
[+|-] [<flags>] [ADD|SUB] [ALL] [QUIET]

Template: FILE/A,FLAGS,ADD/S,SUB/S,ALL/S,QUIET/S

Purpose: To change the protection bits of a file.

Path: C:PROTECT

Specification:

All files have a series of protection bits stored with them which control their attributes. These bits can be altered to indicate the type of file and the file operations permitted. PROTECT is used to set or clear the protection bits of a file.

The protection bits are represented by letters:

r	The file can be read.
w	The file can be written to (altered).
e	The file is executable (a program).
d	The file can be deleted.
s	The file is a script.
p	The file is a pure command and can be made resident.
a	The file has been archived.

To see the protection bits associated with a file, use the LIST command. The protection field is displayed with set (on) bits shown by their letters and clear (off) bits shown by hyphens. For instance, a file that is readable, writable and deletable, will have ----rw-d in the protection field.

To specify the entire protection field at once, simply give the letters of the bits you want set as the FLAGS argument, without any other keywords. The named bits will be set, and all the others will be cleared.

The symbols + and - (or the equivalent keywords ADD and SUB) are used to control specific bits without affecting the state of unspecified bits. Follow + or - with the letter(s) of the bit(s) to set or clear, respectively, and only those bits will be changed. Don't put a space after the symbol or between the letters. The order of the letters does not matter. ADD and SUB work similarly, but there must be a space between the keyword and the letter(s). You cannot both set and clear bits in the same command.

The ALL option adds or removes the specified protection bits from all the files in the specified directory. The QUIET option suppresses the screen output.

Examples:

```
1> PROTECT DF0:Memo +rw
```

sets only the protection bits r (readable) and w (writable) to the file Memo on DF0:. No other protection bits are changed.

```
1> PROTECT L:#? e SUB
```

clears the e (executable) protection bit from all the files in the L: directory.

```
1> PROTECT Work:Paint rwd
```

The protection status of Paint becomes "----rwd".

See Also: LIST

QUIT

Format: QUIT [<return code>]

Template: RC/N

Purpose: To exit from a script file with a specified return code.

Path: Internal

Specification:

QUIT is used to stop the execution of the script upon the specified return code. The default return code is zero. It is recommended that you use the standard return code values of 5, 10 and 20.

Example:

```
ASK "Do you want to stop now?"
IF WARN
QUIT 5
ENDIF
ECHO "OK"
ECHO "The script is continuing."
```

P

Q

If you press Y at the prompt, the script will be aborted, as WARN is equal to a return code of 5. If you press N or press Return:

OK

The script is continuing.

will be displayed in the Shell window.

RELABEL

Format: RELABEL [DRIVE] <drive> [NAME] <name>

Template: DRIVE/A,NAME/A

Purpose: To change the volume name of a disk.

Path: C:RELABEL

Specification:

RELABEL changes the volume name of the disk in the given drive to the <name> specified. Volume names are set initially when you format a disk.

If you have a floppy disk system with only one disk drive, be sure to specify the disks by volume name, instead of drive name.

Examples:

```
1> RELABEL Workbench2.0: My2.0Disk
```

changes the name of the Workbench2.0 disk to My2.0Disk. Notice that you don't need the colon after the second name.

```
1> RELABEL DF2: DataDisk
```

changes the name of the disk in DF2: to DataDisk.

REMRAD

Format: REMRAD [<drive>] [FORCE]

Template: DRIVE, FORCE

Purpose: To remove the recoverable ramdrive.device.

Path: C:REMRAD

Specification:

If you want to remove the recoverable ramdrive.device (usually mounted as RAD:) from memory, and you do not want to turn the system off, you can use the REMRAD command. If you have mounted more than one recoverable ramdrive.device, use the DRIVE specification.

REMRAD commands the ramdrive.device to delete all of its files and become inactive. The next time the Amiga is rebooted, the ramdrive.device is removed from memory completely. If the device is in use at the time the REMRAD command is given, the operation will abort with a drive in use message. To remove it even if it is in use, you must use the FORCE option.

RENAME

Format: RENAME [{FROM}] <name> [TO|AS] <name> [QUIET]

Template: FROM/A/M, TO = AS/A, QUIET/S

Purpose: To change the name of a file or directory.

Path: C:RENAME

Specification:

RENAME renames the FROM file or directory with the specified TO name. *FROM* and *TO* must be on the same disk.

The colon before the directory indicates that the directory is in the root directory.

If the name refers to a directory, RENAME leaves the contents of the directory unchanged (the directories and files within that directory keep the same names and contents).

If you rename a directory, or if you use RENAME to give a file another directory name (for example, you rename :Bill/Letter to :Mary/Letter), AmigaDOS changes the position of that directory or file in the filing system hierarchy.

Examples:

```
1> RENAME Work/Prog1 AS :Arthur/Example
```

renames the file Prog1 as Example, and moves it from the Work directory to the Arthur directory. The Arthur directory must exist in the root directory for this command to work.

```
1> RENAME 7.2Fax 8.16Fax 9.22Fax TO Faxes
```

moves the 7.2Fax, 8.16Fax, and 9.22Fax files to the Faxes directory. The Faxes directory must already exist.

RESIDENT

Format: RESIDENT [<resident name>] [<filename>]
[REMOVE] [ADD] [REPLACE] [PURE|FORCE]
[SYSTEM]

Template: NAME,FILE,REMOVE/S,ADD/S,REPLACE/S,
PURE = FORCE/S,SYSTEM/S

Purpose: To display and modify the list of resident commands.

Path: Internal

Specification:

RESIDENT is used to load commands and add them to the resident list maintained by the Shell. This allows the command to be executed without it having to be reloaded from disk each time. This eliminates the time it takes to load the command and reduces memory use when multitasking.

To be made resident, a command should be both *reentrant* and *re-executable*. A reentrant command can properly support independent use by two or more programs at the same time. A re-executable command does not have to be reloaded to be executed again. Commands that have these characteristics are called *pure* and have the p (pure) protection bit set.

LIST the C: directory to check for the presence of the p protection bit to determine which commands are pure.

Many of the commands in the C: directory, as well as the More command in Utilities, are pure commands and can be made resident. If a command does not have its pure bit set, it probably cannot be made resident safely. (Just setting the pure bit does not make a command or program pure).

The REPLACE option is the default option and does not need to be explicitly stated. If RESIDENT is invoked with no options, it lists the programs on the resident list. If no <resident name> is specified (i.e., just a filename is specified), RESIDENT will use the filename portion as the name on the resident list.

NOTE: The full path to the file must be used.

If a <resident name> is specified and RESIDENT finds a program with that name already on the list, it will attempt to replace the command. That <resident name> must then be used to reference the resident version of the command. The replacement will succeed only if the already-resident command is not in use.

To override REPLACeMENT and make several versions of a command resident simultaneously, use the ADD option, giving a different <resident name> for each version loaded.

If the SYSTEM option is specified, the command will be added to the system portion of the resident list. Any commands added to the resident list with the SYSTEM option cannot be removed. To list SYSTEM files on the RESIDENT list, you must specify the SYSTEM option.



The PURE option forces RESIDENT to load commands which are not marked as pure (i.e., they do not have their pure bit set), and can be used experimentally to test the pureness of other commands and programs.

Use the PURE option with caution. Remember that in order for a command to be made RESIDENT, it must be both reentrant and re-executable. Although it is unlikely, some of your programs may be pure enough to be fully reentrant and usable by more than one process at the same time. Other programs may not be fully reentrant but may be pure enough to be re-executable. Such commands can be made RESIDENT, but you must be extremely careful to use the command in only one process at a time.

The availability of Internal commands can also be controlled with RESIDENT. To deactivate an Internal command (for instance if an application has its own command of the same name), use RESIDENT <Command> REMOVE. AmigaDOS will no longer recognize the Internal command. The AmigaDOS command can be reactivated with the REPLACE option.

Examples:

```
1> RESIDENT C:COPY
```

makes the COPY command resident (replaces any previous version).

```
1> RESIDENT Copy2 DF1:C/COPY ADD
```

adds another version of COPY to the resident list, under the name Copy2.

```
1> RESIDENT Xdir DF1:C/Xdir PURE
```

makes an experimental, non-pure version of the DIR command resident.

1> RESIDENT CD REMOVE

makes the Internal CD command unavailable.

1> RESIDENT CD REPLACE

restores the CD command to the system.

See Also: PROTECT

RUN

Format: RUN <command> [+ {<command>}]

Template: COMMAND/F

Purpose: To execute commands as background processes.

Path: C:RUN

Specification:

RUN is used to launch **background processes**. A background process does not open its own window for input or output and does not take over the parent Shell.

RUN attempts to execute the <command> and any arguments entered on the command line. You can RUN multiple commands by separating them with plus signs (+). If you press Return after a plus sign, RUN will interpret the next line as a continuation of the same command line.

To allow the closing of the Shell window in which the process was started, redirect the output of RUN with RUN >NIL:
<command>.

A new background Shell has the same search path and command stack size as the Shell from which RUN was given.

You can RUN commands stored on the resident list. For speed, resident commands are checked before commands in the command path. A Shell started with RUN NEWSHELL still uses the default startup file, S:Shell-startup.

Examples:

```
1> RUN COPY Text PRT: +  
DELETE Text +  
ECHO "Printing finished"
```

prints the Text file by copying it to the printer device, deletes it, then displays the given message. Plus signs are used to concatenate the command lines.

```
1> RUN EXECUTE Comseq
```

executes, in the background, all the commands in the file Comseq.

SAY

Format: SAY [-m] [-f] [-r] [-n] [-s <n>] [-p <n>]
[-x <filename>]

Template: -m/S,-f/S,-r/S,-n/S,-s/K/N,-p/K/N,-x/K

Purpose: To speak phrases or files through the Amiga.

Path: SYS:Utilities/Say

Specification:

SAY utilizes the Amiga's speech capabilities. It supports the same options as when run through the Workbench, except that when run through the Shell, you can specify a file and its contents will be spoken. The options are:

- m Specifies a male voice.
- f Specifies a female voice.
- r Specifies a robot voice.
- n Specifies a natural voice.

- s <n> Type the -s option, followed by a number from 40 to 400 to control the speed of the voice. Do not put a space between the -s and the number.
- p <n> Type the -p option, followed by a number from 65 to 320 to control the pitch of the voice.
- x <filename> Type the -x option followed by a filename, and the Amiga will "read" the contents of that file.

Do not forget the hyphen before each alphabetical option.

Example:

```
1> SAY -m -s125 -p65 -x s:startup-sequence
```

The Amiga will read the contents of the startup-sequence file in a male voice at a moderately-paced speed.

SCREENMODE

Format: SCREENMODE [FROM <filename>] [EDIT] [USE]
[SAVE]

Template: FROM,EDIT/S,USE/S,SAVE/S

Purpose: To select a display mode.

Path: SYS:Prefs/ScreenMode

Specification:

SCREENMODE without any arguments, or with the EDIT argument, opens the ScreenMode editor. The FROM argument lets you specify a file to open. This must be a file that was previously saved with the Save As menu item of the ScreenMode editor. For instance, if you have saved a special configuration of the ScreenMode editor to a file in the Presets drawer, you can use the FROM argument to open that file. If the USE switch is also given, the editor will not be opened, but the settings in the FROM file will be used. If the SAVE switch is given, the editor will not open, but the settings in the FROM file will be saved.

Example:

```
1> SCREENMODE Prefs/Presets/ScreenMode.Hires USE
```

You will be prompted to close all non-drawer windows, and the system will reset and use the settings saved in the ScreenMode.Hires file. The editor window will not open. When the system is rebooted, the display mode will return to the last selection saved.

SEARCH

Format: SEARCH [FROM] <name|pattern>
[SEARCH|NAME] <string|pattern> [ALL]
[NONUM] [QUIET] [QUICK] [FILE] [PATTERN]

Template: FROM/A/M,SEARCH/A,ALL/S,NONUM/S,
QUIET/S,QUICK/S,FILE/S,PATTERN/S

Purpose: To look for the specified text string in the files of the specified directory or directories.

Path: C:SEARCH

Specification:

SEARCH looks through all the files in the FROM directory for the given SEARCH string. (The FROM and SEARCH keywords are optional.) If the ALL switch is given, SEARCH also looks through all the subdirectories of the FROM directory. SEARCH displays the name of the file being searched and any line that contains the text sought. You must place quotation marks around any search text containing a space. The search is case-indifferent (capitalization is ignored).

The options are as follows:

- | | |
|-------|---|
| NONUM | Line numbers are not printed with the strings. |
| QUIET | Searches quietly; filenames being searched are not displayed. |
| QUICK | Uses a more compact output format. |

FILE Looks for a file by the specified name, rather than for a string in the file.

PATTERN Uses pattern matching in the search.

SEARCH leaves a 0 in the condition flag if the object is found, and a 5 (WARN) otherwise. This makes it useful in scripts. To abandon the search of the current file and continue to the next file, if any, type Ctrl-D. SEARCH is aborted when a Ctrl-C is typed.

Examples:

```
1> Search DEVS: alternative
  (dir)
    Keymaps (dir)
    Printers (dir)
  clipboard.device..
  MountList..
```

```
14 /* This is an example of an alternative type of non-filing device mount.
  narrator.device..
  parallel.device..
  printer.device..
  serial.device..
  system-configuration..
```

searches through the DEVS: directory for the word alternative. It is found on line 14 of the MountList file.

```
1> SEARCH Universe: "Intelligent life" ALL
```

searches for Intelligent life (or intelligent life) in every file on the volume Universe:.

```
1> SEARCH Work: #?.source SEARCH Progtest.c?? FILE PATTERN
```

locates all Progtest.c files with a two-character suffix in directories ending in .source in the Work volume.

SERIAL

Format: SERIAL [FROM <filename>] [EDIT] [USE]
[SAVE]

Template: FROM,EDIT/S,USE/S,SAVE/S

Purpose: To set the specifications for communication through the serial port.

Path: SYS:Prefs/Serial

Specification:

SERIAL without any arguments or with the EDIT argument opens the Serial editor. The FROM argument lets you specify a file to open. This must be a file that was previously saved with the Save As menu item of the Serial editor. For instance, if you have saved a special configuration of the Serial editor to a file in the Presets drawer, you can use the FROM argument to open that file. If the USE switch is also given, the editor will not open, but the settings in the FROM file will be used. If the SAVE switch is given, the editor will not open, but the settings in the FROM file will be saved.

Example:

```
1> SERIAL Prefs/Presets/Serial.9600 USE
```

loads and uses the specifications saved in the Serial.9600 file. If the system is rebooted, the last saved settings will take effect.

SET

Format: SET [<name>] [<string>]

Template: NAME,STRING/F

Purpose: To set a local variable.

Path: Internal

Specification:

SET with <name> and <string> arguments creates a new environment variable. The first word after SET is taken as the <name>. Everything else on the command line is taken as the <string> argument. Quotation marks are not required.

SET with no arguments list the current local variables.

An environment variable created with SET is local to the Shell in which it was created. If you create a new Shell with the NEWSHELL command, that Shell will also recognize any variables created in its parent Shell. However, if you create a new Shell with the Execute Command menu item or by opening the Shell icon, variables created with SET will not be recognized.

You can call environment variables in a script or on a command line by placing a dollar sign (\$) in front of the variable name.

To remove a local variable definition, use the UNSET command.

Examples:

```
1> SET origin This process launched from icon
```

creates the local variable origin which stores a reminder that a Shell was invoked from an icon rather than a NEWSHELL.

```
1> ECHO $origin  
This process launched from icon
```

See Also: GET, UNSET

SETCLOCK

Format: SETCLOCK LOAD|SAVE|RESET

Template: LOAD/S,SAVE/S,RESET/S

Purpose: To set or read the battery backed-up hardware clock.

Path: C:SETCLOCK

Specification:

The battery backed-up clock keeps the time even when the Amiga is turned off. Amiga 500s do not have battery backed-up clocks, unless an A501 RAM expansion cartridge has been installed.

SETCLOCK SAVE sets the date and time of the battery backed-up hardware clock from the current system time (saved with the Time editor or with the DATE command). SETCLOCK SAVE is typically used after a DATE command.

SETCLOCK LOAD sets the current system time from the battery backed-up clock. It is typically included in the Startup-sequence to automatically load the correct time when the Amiga is booted.

The RESET option resets the clock completely. This may be necessary if a poorly written program that does not follow the rules turns the clock off or sets the test bit of the clock.

Example:

```
1> DATE 17-Aug-92 05:45:54
1> SETCLOCK SAVE
```

saves the date, August 17, 1992, and the time, 5:45 a.m., to the battery backed-up hardware clock. That clock keeps time even when the Amiga is powered off. When the system is booted, the SETCLOCK LOAD command in the Startup-sequence sets the system clock with the time saved in the hardware clock.

See also: DATE

SETDATE

Format: SETDATE <file|pattern> [<weekday>]
[<date>] [<time>] [ALL]

Template: FILE/A,WEEKDAY,DATE,TIME,ALL/S

Purpose: To change a file or directory's timestamp.

Path: C:SETDATE

Specification:

SETDATE changes the **timestamp** (date and time of creation or last change) of a file or directory. SETDATE <file> changes the date/time of the file to the current system date/time.

SETDATE does not affect the software or hardware clocks.

The output of the DATE command may be used as input to SETDATE.

Examples:

1> SETDATE TestFile

changes the date and time associated with TestFile to the current date and time.

1> SETDATE TestFile 16-09-89 15:25:52

change the date and time associated with TestFile to September 16, 1989, 3:25 p.m.

See Also: DATE

SETENV

Format: SETENV [<name>] [<string>]

Template: NAME,STRING/F

Purpose: To set a global variable.

Path: Internal

Specification:

SETENV with <name> and <string> arguments creates a new global environment variable. The first word after SETENV is taken as the <name>. Everything else on the command line is taken as the <string> argument. Quotation marks are not required. Comments (statements prefaced with a semicolon (;)) are not allowed on a SETENV command line. This includes command lines that are part of a script.

SETENV with no arguments list the current global variables. Global variables are stored in ENV: and are used by all processes. If a local variable (defined by SET) and a global variable share the same name, the local variable will be used.

Environment variables are called by scripts or other commands by including a dollar sign (\$) in front of the variable name.

To remove a global variable definition, use the UNSETENV command.

Examples:

```
1> SETENV Editor Extras2.0:Tools/MEmacs
```

creates the environment variable Editor which can be used with the More utility. This specifies the editor as being MEmacs, located in the Tools drawer of the Extras2.0 disk. The variable Editor will be available in any Shell.

```
1> SETENV Editor C:ED
```

same as above, only the editor specified is the editor ED.

```
1> ECHO $Editor  
C:ED
```

See Also: GETENV, UNSETENV

SETFONT

Format: SETFONT <size> [SCALE] [PROP]
[ITALIC] [BOLD] [UNDERLINE]

Template: NAME/A,SIZE/A,SCALE/S,PROP/S,
ITALIC/S,BOLD/S,UNDERLINE/S

Purpose: To change the Shell font.

Path: C:SETFONT

Specification:

SETFONT lets you change the font used in a particular Shell window, overriding the System Default Text setting specified in the Font editor. SETFONT is only effective in the window in which it is invoked.

You must specify both a font name and a size when using the SETFONT command. The other options are:

SCALE	Enables bitmap font scaling.
PROP	Allows proportional fonts.
ITALIC	The font will be italic.
BOLD	The font will be boldface.
UNDERLINE	The font will be underlined.

Invoking SETFONT will clear the Shell window of its current contents and display a new prompt, in the new font, at the top of the window.

Example:

```
1> SETFONT Topaz 13 BOLD UNDERLINE
```

The Shell window will clear, and the new prompt will be in 13 point Topaz, underlined and boldface.

SETMAP

Format: SETMAP <keymap>

Template: KEYMAP/A

Purpose: To change the keymap used by the Amiga.

Path: SYS:System/SetMap

Specification:

SETMAP specifies the keymap used by the Amiga. The available files are listed below:

Keymap	Keyboard
cdn	French Canadian
chl	Swiss French
ch2	Swiss German
d	German
dk	Danish
e	Spanish
f	French
gb	Great Britain
i	Italian
is	Icelandic
n	Norwegian
s	Swedish
usa0	(For programs developed before V1.0)
usa	American
usa2	Dvorak

To have the system always use a different keymap, add the SETMAP command to your Startup-sequence file.

Example:

To change to a French Canadian keymap, type:

```
1> SETMAP cdn
```

The keymap file must be in DEVS:Keymaps for SetMap to find it.

SETPATCH

Format: SETPATCH [NOCACHE] [QUIET]

Template: NOCACHE/S,QUIET/S

Purpose: To make ROM patches in system software.

Path: C:SETPATCH

Specification:

SETPATCH installs temporary modifications to the operating system. If needed, it must be run at the beginning of the Startup-sequence file. Updated versions of SETPATCH will be made available when necessary as AmigaDOS development continues.

The NOCACHE option disables data cache on some third-party accelerator cards. If you have been experiencing system failures and you have a third-party accelerator card in your machine, you should try adding the SETPATCH NOCACHE command to the beginning of your Startup-sequence.

The QUIET switch suppresses any screen output.

SKIP

Format: SKIP [<label>] [BACK]

Template: LABEL,BACK/S

Purpose: To skip to a label when executing script files.

Path: Internal

Specification:

SKIP is used in scripts to allow you to skip ahead in the script to a <label> defined by a LAB statement. If no <label> is specified, SKIP jumps to the next LAB statement.

SKIP always searches forward from the current line of the file. However, when the BACK option is used, SKIP starts searching for the label from the beginning of the file. This allows SKIPs to points prior to the SKIP command.

You can only SKIP back as far as the last EXECUTE statement. If there are no EXECUTE statements in a script, you will SKIP back to the beginning of the file.

If SKIP does not find the label you specified, the command sequence terminates and the message Label <label> not found by Skip is displayed.

Example:

Assume you have the following script, called CheckFile:

```
.KEY name
IF exists <name>
    SKIP message
ELSE
    ECHO "<name> is not in this directory."
ENDIF
LAB message
    ECHO "The <name> file does exist."
```

You can run the script by typing:

```
1> EXECUTE CheckFile Document
```

If the Document file exists in the current directory, the execution of the script will skip ahead to the LAB command. The message The Document file does exist will be displayed in the Shell window.

If the Document file is not in the current directory, the execution of the script will jump to the line after the ELSE statement, and the message Document is not in this directory will be displayed.

See also: EXECUTE, LAB

SORT

Format: SORT [FROM] <file|pattern> [TO] <filename>
[COLSTART <n>] [CASE] [NUMERIC]

Template: FROM/A,TO/A,COLSTART/K,CASE/S,
NUMERIC/S

Purpose: To alphabetically sort the lines of a file.

Path: C:~\SORT

Specification:

SORT will sort the FROM file alphabetically, line-by-line, sending the sorted results to the TO file. SORT assumes the file is a normal text file in which lines are separated by Returns or line feeds. SORT normally disregards capitalization. If the CASE switch is given, capitalized items will be output first.

The COLSTART keyword allows you to specify the character column at which the comparison will begin. SORT compares the lines from that point on, and comparison will wrap around to the beginning of the line if the lines being compared match to the end.

When the NUMERIC option is specified, the lines are interpreted as numbers from the first column rightward, stopping at the first non-numeric character. Lines not beginning with numbers are treated as 0. The lines are output in numerical order. If the CASE switch is given with NUMERIC, CASE is ignored.

Example:

```
1> SORT DF0:Glossary DF0:Glossary.alpha
```

sorts the lines in the Glossary file, arranges them alphabetically, and outputs them to a new file called Glossary.alpha. The case of the words is disregarded.

STACK

Format: STACK [<n>]

Template: SIZE/N

Purpose: To display or set the stack size within the current Shell.

Path: Internal

Specification:

When you run a program, it uses a certain amount of **stack**, a special area in memory allocated for the program. The stack required for a program should be given in the program's documentation. However, if a program causes system failure, you may wish to experiment with various stack sizes.

Commands that perform operations that consist of multiple levels may require additional stack space.

Stack sizes generally range from 4000 to 25000 bytes. If the stack size is too small, a system failure may occur. Too large of a stack size may take too much memory away from other system functions.



If you run out of stack space, you may receive a Software Failure message. If you have altered the stack for the program that caused the Software Failure message, try increasing the stack size.

STATUS

Format: STATUS [<process>] [FULL] [TCB] [CLI|ALL]
[COMMAND <command>]

Template: PROCESS/N,FULL/S,TCB/S,CLI= ALL/S,
COM= COMMAND/K

Purpose: To list information about Shell/CLI processes.

Path: C:STATUS

Specification:

STATUS without any arguments lists the numbers of the current Shell/CLI processes and the program or command, if any, running in each. The <process> argument specifies a process number, and STATUS will only give information about that process.

For information on the stacksize, global vector size, priority, and current command for each process, use the FULL keyword. The TCB keyword is similar, but omits the command information.

With the COMMAND option, you can tell STATUS to search for a command. STATUS then scans the Shell list, looking for the specified <command>. If the command is found, the Shell number is output, and the condition flag is set to 0. Otherwise the flag is set to 5 (WARN). This is useful in script files.

Examples:

```
1> STATUS 1
```

```
Process 1: Loaded as command: status
```

```
1> STATUS 1 FULL
```

```
Process 1: stk 4000, gv 150, pri 0 Loaded as command: status
```

```
1> STATUS >RAM:xyz COMMAND= COPY
```

```
1> BREAK <RAM:xyz >NIL: ?
```

sends a break to the process executing COPY.

See Also: BREAK

TIME

Format: TIME [EDIT]

Template: EDIT/S

Purpose: To set the system clock.

Path: SYS:PREFS/TIME

Specification:

TIME without any arguments or with the EDIT argument opens the Time editor.

Example:

```
1> TIME
```

TYPE

Format: TYPE {<file|pattern>} [TO <name>] [OPT H|N|HEX] [NUMBER]

Template: FROM/A/M,TO/K,OPT/K,HEX/S,NUMBER/S

Purpose: To display a text file.

Path: C:TYPE

Specification:

TYPE will output the contents of the named file to the current window, if no destination is given, or to a specified output file. If more than one filename is specified, and the TO keyword is not used, the filenames will be typed in sequence.

The OPT H and OPT N options are also available by the HEX and NUMBER keywords, respectively. The HEX option causes the file to be typed as columns of hexadecimal numbers, with an ASCII character interpretation column. This is useful for analyzing object files. The NUMBER option will number the lines as they are output.

To pause output, press the Space bar. To resume output, press Backspace, Return, or Ctrl-X. To stop output, press Ctrl-C (**BREAK is displayed).

Example:

```
1> TYPE DEVS:MountList
```

The contents of the MountList file in the DEVS: directory will be displayed on the screen.

UNALIAS

Format: UNALIAS [<name>]

Template: NAME

Purpose: To remove an alias.

Path: Internal

Specification:

UNALIAS removes the named alias from the alias list.

With no arguments, UNALIAS lists the current aliases.

See also: ALIAS

UNSET

Format: UNSET [<name>]

Template: NAME

Purpose: To remove a local variable.

Path: Internal

Specification:

UNSET removes the named local variable from the variable list for the current process.

With no arguments, UNSET lists the current variables.

See also: SET

T**U**

UNSETENV

Format: UNSETENV [<name>]
Template: NAME
Purpose: To remove a global variable.
Path: Internal
Specification:

UNSETENV removes the named global variable from the current variable list. With no arguments, UNSETENV lists the current variables.

See also: SETENV

VERSION

Format: VERSION [<library|device|file>]
[<version #>] [<revision #>] [<unit #>]
[FILE] [INTERNAL] [RES] [FULL]
Template: NAME,VERSION,REVISION,UNIT,FILE/S,
INTERNAL/S,RES/S,FULL/S
Purpose: To find software version and revision numbers.
Path: C:VERSION
Specification:

VERSION finds the version and revision number of a library, device, command, or Workbench disk. VERSION can also test for a specific version/revision and set the condition flags if the version/revision is greater. This is useful in scripts.

VERSION with no <library|device|file> argument prints the Kickstart version number and the Workbench version number and sets the environment variables. If a name is specified, version attempts to open the library, device, drive, or file and read the version information. You can get the version of the filesystem by specifying a drive name, such as DF0: or DH0:.

When a <version #> (and possibly a <revision #>) is specified, VERSION sets the condition flag to 0 if the version (and revision) number of the Kickstart, library, or device driver is greater than or equal to the specified values. Otherwise, the flag is set to 5 (WARN). (If a revision number is not specified, no comparison on the revision number is performed.)

The <unit #> option allows you to specify a unit number other than 0. This may be necessary for accessing multi-unit devices.

The FILE option forces VERSION to inspect the object as a file, even if it is a library or device. The INTERNAL and RES options let you get the version of the Internal and Resident commands, respectively. Built-in Shell commands will have the same version string as the Shell itself. INTERNAL can also be used to find the version of a RAM module (library or drive) without opening the device or library. The FULL option prints out the complete version string, including the date.

Examples:

```
1> VERSION
```

```
Kickstart version 36.202 Workbench version 36.77
```

```
1> VERSION Prefs/Font
```

```
Prefs/Font version 36.191
```

WAIT

Format: WAIT [<n>] [SEC|SECS] [MIN|MINS] [UNTIL
<time>]

Template: /N,SEC = SECS/S,MIN = MINS/S,UNTIL/K

Purpose: To wait for the specified time.

Path: C:WAIT

Specification:

WAIT is used in command sequences or after RUN to wait for a certain period, or to wait until a certain time. Unless you specify otherwise, the waiting period is one second. The <n> argument specifies the number of seconds (or minutes, if MINS is given) to wait. Use the keyword UNTIL to wait until a particular time of the day, given in the format HH:MM.

Examples:

1> WAIT 10 MINS

waits ten minutes.

1> WAIT UNTIL 21:15

waits until 9:15 p.m.

WBPATTERN

Format: WBPATTERN [FROM <filename>] [EDIT] [USE]
[SAVE] [WORKBENCH] [WINDOW]

Template: FROM,EDIT/S,USE/S,SAVE/S,WORKBENCH/S,
WINDOW/S

Purpose: To create background patterns for the Workbench
and windows.

Path: SYS:Prefs/WBPattern

Specification:

WBPATTERN with no arguments or with the EDIT argument opens the WBPattern editor.

The FROM argument must be used in combination with a WORKBENCH or WINDOW switch. (You can use more than one switch.) This allows you to specify a particular pattern to be used in the designated area(s) of the screen. The FROM file must be one that was previously saved with the Save As menu item of the WBPattern editor's Project menu. Even if the pattern in the FROM file was originally saved for one area of the screen, it can be used in a different area of the screen by specifying the appropriate switch. For instance, if the FROM file was created when you saved a pattern for the Workbench, that pattern can be used in the windows by specifying the WINDOW switch after the filename.

If you specify the USE option, the pattern will be loaded into the appropriate area and used, just as if you had opened the WBPattern editor, selected the appropriate radio button, created the pattern, and selected the Use gadget. If you specify the SAVE option, that pattern will be saved.

If you do not specify USE or SAVE, EDIT is assumed, and the WBPattern editor is opened. If a FROM file and a WORKBENCH or WINDOW switch is specified, the Font editor will open, the pattern saved in the FROM file will be displayed, and the appropriate radio button will be selected. If no switch is specified with the FROM file, the editor will display the last used pattern.

Examples:

```
1> WBPATTERN FROM SYS:Prefs/Presets/Diamonds WORKBENCH SAVE
```

loads and saves the pattern saved in the Diamond file as the background pattern for the Workbench.

```
1> WBPATTERN FROM SYS:Prefs/Presets/Dots WINDOWS
```

opens the WBPattern editor. The pattern saved in the Dots file will appear in the magnified view, and the Windows radio button will be selected. You must select the Save, Use, or Cancel gadget to proceed.

WHICH

Format: WHICH <command> [NORES] [RES] [ALL]

Template: FILE/A,NORES/S,RES/S,ALL/S

Purpose: To search the command path for a particular item.

Path: C:WHICH

Specification:

WHICH lets you find a particular command, program, or directory by entering its name. If the named item is in the search path, WHICH displays the complete path to that item. WHICH lists resident commands as RESIDENT and internal commands as INTERNAL.

Normally, WHICH searches the resident list, the current directory, the command path(s), and the C: directory. The condition flag is set to 5 (WARN) if the file is not found.

If the NORES option is specified, the resident list is not searched. If the RES option is specified, only the resident list is searched.

The ALL switch causes the search to continue through the full search path, even after one or more instances of the named item have been found and listed. This insures that all versions of a command or program are found. It can, however, lead to multiple listings of the *same* command, if that command is reached by more than one route (e.g., C: and the current directory).

Examples:

```
1> WHICH avail  
C:avail
```

```
1> WHICH C:  
Workbench2.0:C
```

```
1> WHICH alias  
INTERNAL alias
```

WHY

Format: WHY

Template: (none)

Purpose: To print an error message that explains why the previous command failed.

Path: Internal

Specification:

Usually when a command fails the screen displays a brief message. This message typically includes the name of the file (if that was the problem) but does not go into detail.

For example, the message Can't open <filename> may appear. This could happen for a number of reasons—AmigaDOS may not be able to locate the file, the file is of the wrong type, or there was insufficient disk space or RAM for the operation requested.

If the reason is not immediately obvious, enter WHY to get a more complete explanation.

Examples:

1> COPY DF0:

Bad arguments

1> WHY

Last command failed because required argument missing

The WHY message points to the error: a destination for the COPY was not given.

Error Messages

This section lists the possible AmigaDOS errors, along with probable causes and suggestions for recovery. Programmer errors are boxed.

Error	Probable Cause	Recovery Suggestion
103 Not enough memory	Not enough memory in your Amiga to carry out the operation.	Close any unneeded windows and applications, then reissue the command. If it still doesn't work, try rebooting. It may be that you have enough memory but it has become fragmented. It is possible that you may need to add more RAM to your system.
104 Process table full	There is a limit to the number of possible processes.	Stop one or more tasks.
114 Bad template	Incorrect command line.	Consult the documentation for the correct command format.
115 Bad number	The program was expecting a numerical argument.	Consult the documentation for the correct command format.
116 Required argument missing	Incorrect command line.	Consult the documentation for the correct command format.
117 Argument after '=' missing	Incorrect command line.	Consult the documentation for the correct command format.
118 Too many arguments	Incorrect command line.	Consult the documentation for the correct command format.
119 Unmatched quotes	Incorrect command line.	Consult the documentation for the correct command format.
120 Argument line invalid or too long	Your command line is incorrect or contains too many arguments.	Consult the documentation for the correct command format.
121 File is not executable	You misspelled the command name, or the file may not be a loadable (program or script) file.	Retype the filename and make sure that the file is a program file. Remember, in order to execute a script, either the s bit must be set or the EXECUTE command must be used.

122 Invalid resident library	You are trying to use commands with a previous version of AmigaDOS, e.g. Version 2.0 commands with Version 1.3.	Reboot with the current version of AmigaDOS.
202 Object is in use	The specified file or directory is already being used by another application. If an application is reading a file, no other program can write to it, and vice versa.	Stop the other application that is using the file or directory, and reissue the command.
203 Object already exists	The name that you specified already belongs to another file or directory.	Use another name, or delete the existing file or directory, and replace it.
204 Directory not found	AmigaDOS cannot find the directory you specified. You may have made a typing or spelling error.	Check the directory name (use DIR if necessary). Reissue the command.
205 Object not found	AmigaDOS cannot find the file or device you specified. You may have made a typing or spelling error.	Check the filename (use DIR) or the device name (use INFO). Reissue the command.
206 Invalid window description	This occurs when specifying a window size for a Shell, ED, or ICONX window. You may have made the window too big or too small, or you may have omitted an argument. This error also occurs with the NEWSHELL command, if you supply a device name that is not a window.	Reissue the window specification.
209 Packet request type unknown	You have asked a device handler to attempt an operation it cannot do. For example, the console handler cannot rename anything.	Check the request code passed to device handlers for the appropriate request.
210 Object name invalid	There is an invalid character in the filename or the filename is too long. Remember, filenames cannot be longer than 30 characters and cannot contain control characters.	Retype the name, being sure not to use any invalid characters or exceed the maximum length.

211 Invalid object lock	You have used something that is not a valid lock.	Check that your code only passes valid locks to AmigaDOS calls that expect locks.
212 Object not of required type	You may have specified a filename for an operation that requires a directory name, or vice versa.	Consult the documentation for the correct command format.
213 Disk not validated	If you have just inserted a disk, the disk validation process may be in progress. It is also possible that the disk is corrupt.	If you've just inserted the disk, wait for the validation process to finish. This may take less than a minute for a floppy disk or up to several minutes for a hard disk. If the disk is corrupt, it cannot be validated. In this case, try to retrieve the disk's files and copy them to another disk. You may have to use DISKDOCTOR.
214 Disk is write-protected	The plastic tab is in the write-protect position.	If you're certain you want to write to that particular disk, remove the disk, move the tab, and reinsert the disk. Otherwise, use a different disk.
215 Rename across devices attempted	RENAME only changes a filename on the same volume. You can use RENAME to move a file from one directory to another, but you cannot move files from one volume to another.	Use COPY to copy the file to the destination volume. Delete it from the source volume, if desired. Then use RENAME.
216 Directory not empty	This error occurs if you attempt to delete a directory that contains files or subdirectories.	If you are sure you want to delete the complete directory, use the ALL option of DELETE.
217 Too many levels	You've exceeded the limit of 15 soft links.	Reduce the number of soft links.
218 Device (or volume) not mounted	If the device is a floppy disk, it has not been inserted in a drive. If it is another type of device, it has not been mounted with MOUNT. It is also possible that you have made a typing error when specifying the device name.	Insert the correct floppy disk, check the spelling of the device name, mount the device, or revise your MountList file.

219	Seek error	You have attempted to call SEEK with invalid arguments.	Make sure that you only SEEK within the file. You cannot SEEK outside the bounds of the file.
220	Comment is too long	Your filenote has exceeded the maximum number of characters (79).	Use a shorter filenote.
221	Disk is full	There is not enough room on the disk to perform the requested operation.	Delete some unnecessary files or directories, or use a different disk.
222	Object is protected from deletion	The d (deletable) protection bit of the file or directory is clear.	If you are certain that you want to delete the file or directory, use PROTECT to set the d bit or use the FORCE option of DELETE.
223	File is write protected	The w (writeable) protection bit of the file is clear.	If you are certain that you want to overwrite the file, use PROTECT to set the w bit.
224	File is read protected	The r (readable) protection bit of the file is clear.	Use PROTECT to set the r bit of the file.
225	Not a valid DOS disk	The disk in the drive is not an AmigaDOS disk, it has not been formatted, or it is corrupt.	Check to make sure you are using the correct disk. If you know the disk worked before, use DISKDOCTOR or another disk recovery program to salvage its files. If the disk has not been formatted, use FORMAT to do so.
226	No disk in drive	The disk is not properly inserted in the specified drive.	Insert the appropriate disk in the specified drive.
232	No more entries in directory	This indicates that the AmigaDOS call EXNEXT has no more entries in the directory you are examining.	Stop calling EXNEXT.
233	Object is soft link	You tried to perform an operation on a soft link that should only be performed on a file or directory.	AmigaDOS uses Action_ReadLink to resolve the soft link and retries the operation.

Chapter 9. Editors

This chapter describes how to use the three text editors supplied with your Amiga: ED, MEmacs, and EDIT. Each editor has the basic functionality of a word processor but does not support style formatting options, such as italics, page numbering, or different fonts.

ED is easy-to-use and is suitable for editing scripts, Startup-sequences, MountLists, and other simple files. MEmacs is more powerful, allows editing of more than one file at a time, and has extensive text-manipulation options. EDIT is a line editor designed for automated editing of files, particularly binary files or files that are larger than available memory.

Each editor is explained in this chapter.

ED

ED is a full-screen, ASCII text editor. It allows bi-directional scrolling of text and supports inserting, deleting and moving blocks of text as well as searching for and replacing words or phrases.

ED features menus and function-key support for quick access to all its features. The mouse can be used to perform nearly any operation. The menus are preprogrammed with the most common operations, but they can be reconfigured to suit your personal preferences. All ED functions are accessible through the keyboard.

NOTE: ED does not accept source files containing binary code. To edit these types of files, use EDIT or MEmacs.

New Features of ED

If you are a current user of ED, you should read this section. If you are a new user, you can skip ahead to the "Starting ED" section on page 9-3.

For Version 2.0, several enhancements have been made to ED:

- ED is now fully mouse-controllable, including cursor positioning.
- Menus have been added which support both editing and file operations.
- The function keys and menus are completely programmable and support the use of macros (multiple-key commands or multiple commands).
- Shifted function keys are also supported and programmable.
- A file requester makes load/save operations graphically controllable.
- Shifted cursor keys allow for quick movement throughout a file.
- Scrolling speed has been increased.
- You can create configuration script files to customize the ED environment.
- AREXX is supported, and AREXX scripts can be run from within ED.
- The window has a close gadget.

Starting ED

You must start ED through a Shell or with the Execute Command menu item. The correct Format and Template are shown below:

Format: ED [FROM] <filename> [SIZE <n>] [WITH]
[WINDOW] [TABS] [WIDTH] [HEIGHT]

Template: FROM/A,SIZE/N,WITH/K,WINDOW/K,TABS/
N,WIDTH=COLS/N,HEIGHT=ROWS/N

You must specify a filename with ED, either the name of an existing file you want to edit or the name of a new file you want to create. If the filename cannot be found in the current directory, ED will treat the specified name as the name of a new file. This filename will be used when you save your work. You do not have to specify the FROM keyword.

For example:

```
1> ED Script
```

opens and displays the file Script. If a file named Script cannot be found in the current directory, a blank ED window opens and displays the message Creating new file.

Because the file is read into memory, there is a limit to the size of the file you can edit with ED. The default size of the text buffer ED uses to hold the file is 40,000 bytes. The SIZE option allows you to change the size of the buffer. For example:

```
1> ED Script SIZE 55000
```

increases the size of the buffer to 55,000 bytes.

When you run ED, the file S:Ed-startup is executed. It is a command file of ED extended mode commands that sets up the default menu assignments. You can edit this file if you want to set up custom menus or preprogrammed function key assignments. Advanced users may want to delete, or rename,

the ED-startup file and create their own customized file of startup options. If the S:Ed-startup file cannot be found, ED will open with an expanded set of menus.

An alternative way to specify startup options is with the WITH argument. WITH allows you to specify the name of an ED command file. This file can contain any sequence of ED extended mode commands (explained on pages 9-10 to 9-19), each on its own line. It can contain the function-key assignments or commands for performing automated editing operations on an existing file.

When the WITH argument is specified, ED executes the entire series of commands included in the file, just as if you had typed them at the keyboard. You can even include Quit commands.

NOTE: Do not include Quit commands in the S:Ed-startup file, or you will not be able to get into ED as it will Quit immediately after opening.

The WINDOW, WIDTH, and HEIGHT arguments allow you to define your terminal type (if you are using a non-Amiga console) or adjust the size of the ED window. The WINDOW argument specifies the console type, such as RAW:0/0/640/256/EdWindow or *. WIDTH and HEIGHT specify the number of characters to display horizontally and vertically. ED supports the console window options explained in Chapter 7.

TABS sets the tab stop interval. This is the number of spaces to the right that the cursor will move when the Tab key is pressed. The default value is 3.

When ED is running, the bottom line of the ED window is the status line. Error messages are displayed there and will remain on the status line until you give another ED command. This is also the line on which you enter extended mode commands.

NOTE: ED always appends a line feed to the end of a file, even if you do not specify one.

Using ED

There are two types of ED commands: immediate and extended. ED opens in immediate mode. In immediate mode, ED executes commands right away. You specify an immediate command by pressing a single key or Ctrl-key combination or by using the mouse. All immediate commands have a corresponding extended version.

To enter extended mode, press Esc. In extended mode, anything you type appears on the command line at the bottom of the window. ED does not execute the command until you press Return. You can type a number of extended commands on a single command line. You can group commands together and have ED repeat them automatically. After ED has executed the command line, it returns to immediate mode.

You can also execute commands through the programmable menus and function keys. Reconfiguring the menus and function keys is simply a matter of assigning a command to the key or menu item of your choice. This is fully explained in the "Customizing ED" section on page 9-21.

In some cases, you must include an argument with a command, such as a number or a **text string**. A text string is a sequence of characters. However, to define where the string begins and ends, you have to use **delimiters**. A delimiter is simply any character except a letter, number, space, semicolon, question mark, or brackets. You can use slashes, symbols, colons, and quotes. For example, some valid strings could be:

```
/DF0:/  
"864 bytes"  
:ECHO "Hello!"
```

A typical use of text strings is to indicate the name of a file to be loaded or saved. However, you can also ask ED to use a file requester. This allows you to view the contents of the drives and directories in your system. (File requesters are fully explained in the "Requesters" section of Chapter 2.)

Be sure to include a space before the question mark and the string.

To invoke the requester after a load or save command, you must place a question mark (?) before the required string argument. Normally, when a command is followed by a string, ED treats the string as the file to be loaded or saved and attempts the operation immediately. However, the question mark tells the command that you want to specify the file through a file requester. You must still specify a string after the question mark, but the string will be the text that appears in the file requester title bar.

For example, SA is an extended command that saves the contents of ED to a specified file. (This is fully explained on page 9-13). If you type:

```
SA !DF0:Docs/List!
```

The file will be saved as List in the Docs directory of DF0:. However, if you want to use a file requester to view the available drives and directories, type:

```
SA ? !Save As?!
```

The question mark tells SA that you want the requester with the words Save As? in the title bar. If for some reason ED is unable to bring up the file requester, a text prompt will appear on the status line, and you can enter the file name there.

Immediate Commands

This section describes the immediate commands. Immediate commands control:

- cursor movement
- text scrolling
- text insertion
- text deletion
- repetition of commands

Moving the Cursor

The cursor can be positioned anywhere in your text by moving the pointer to the desired spot and clicking the selection button. If you prefer to use the keyboard, you can use the cursor keys, Tab, and several Ctrl-key combinations.

To move the cursor one position in any direction, press the appropriate cursor key. If the cursor is on the right edge of the screen, ED scrolls the text to the left so you can see the rest of the line. ED scrolls text vertically one line at a time and horizontally ten characters at a time. You cannot move the cursor off the top or bottom of the file or off the left or right edge of a line. If you try, ED displays a Top of File or Bottom of File message.

Some additional ways to move the cursor are listed below:

Shift-up cursor	Top of the file
Shift-down cursor	Bottom of the file
Shift-left cursor	Left edge of the ED window (regardless of the margin setting)
Shift-right cursor	End of the current line
Ctrl-]	To right-hand edge of current line (if cursor is already there, it is moved to the left-hand edge)
Ctrl-E	Start of the first line on the screen (if cursor is already there, it is moved to the end of the last line on the screen)
Ctrl-T	Start of the next word
Ctrl-R	Space following the previous word
Tab	To the next tab position (multiple of 3)

If your file has more lines than can fit in the ED window, you can scroll through the file vertically. One way to do this is by moving the cursor to the top or bottom of the ED window, and pressing the up or down cursor key. The text will scroll one line at a time. You can move the text in jumps, by pressing:

Ctrl-D	Moves 12 lines down through the file.
--------	---------------------------------------

Ctrl-U	Moves 12 lines up through the file.
--------	-------------------------------------

The commands do not move the cursor position in the window. They just redraw the text in the window with the new line at the cursor position.

If something happens to disturb your screen, such as an alert from another program appearing in the ED window, press Ctrl-V. This will refresh the entire screen.

Inserting Text

While in immediate mode, any characters you type are inserted at the current cursor position, and the cursor is moved to the right. Any characters to the right of the cursor are moved to make room for the new text. If the line is wider than the width of the window, the window scrolls to the left so that you can see what you are typing. If you move the cursor beyond the end of the line, by using the Tab or cursor keys, ED inserts spaces between the end of the line and any new characters you insert.

There is a maximum limit of 225 characters in a line. If you try to add more characters, ED will display a Line Too Long message.

To split the current line at the cursor, press Return. Any text to the left of the cursor will remain on the original line. All text under and to the right of the cursor will move down onto a new line. If the cursor is at the end of the line and you press Return, ED creates a new blank line. In either case, the cursor will appear in the first column of the new line.

Deleting Text

ED has no typeover mode. To replace a word or line, you must delete the existing word(s) and insert new information. You can do this with several keys and key combinations:

Backspace	Deletes the character to the left of the cursor.
Del	Deletes the character under the cursor.
Ctrl-O	If the cursor is over a space, all spaces up to the next character are deleted. If the cursor is over a character, all characters up to the next space are deleted.
Ctrl-Y	Deletes all characters from the cursor to the end of the line.
Ctrl-B	Deletes the entire line.

When text is deleted, any characters remaining on the line shift to the left, and any text beyond the right edge of the screen becomes visible.

Changing Case

You can change the case of text by moving the cursor to the desired location and pressing Ctrl-F. If the letter is lowercase, it will become uppercase, and vice versa. Ctrl-F will not change a non-alphabetic character.

After you press Ctrl-F, the cursor moves to the right. If the next character is a letter, you can press Ctrl-F again to change its case. You can repeat the command until you have changed all the letters on the line. For example, if you had the line:

```
IF <file> <= x
```

and you keep Ctrl-F pressed down, the line would become:

```
if <FILE> <= X
```

Symbols are not affected. If you continue to press Ctrl-F after the last letter on the line, the cursor keeps moving right.

Extended Commands

This section describes the extended commands. Extended commands manage:

- program control
- cursor movement
- text altering
- block control
- searching and exchanging text

To enter extended command mode, press Esc. Subsequent input will appear on the command line at the bottom of the screen. You can correct mistakes with Backspace. To execute the command line, press either Esc or Return. If you press Esc, the editor remains in extended mode, and you can enter another command on the command line. If you press Return, ED is returned to immediate mode.

Extended commands consist of one or two characters. In this section they are shown in uppercase, but they can be entered in either uppercase or lowercase. You can give multiple commands on the same command line by separating them with a semicolon. For instance:

```
T; F /Workbench/
```

moves the cursor to the top of the file, then searches for the next occurrence of the word Workbench.

Program Control

This section provides a specification of the program control commands.

New Project

Esc-NW

Creates a new file, replacing the existing file. The message Edits will be lost – type Y to confirm appears to remind you that this will clear the current file from ED. To save the existing file, press any key except Y, and the command will be aborted.

Open File

Esc-OP

Opens a file. If you want a file requester to appear, type a question mark after the command along with a properly delimited string. For example:

```
OP ? /File?/
```

will bring up a file requester with File? in the title bar. Use the requester to select a file to be loaded into ED.

You can also specify the file to be opened by entering the path to the file as the string. For example:

```
OP /S:Startup-sequence/
```

will load the Startup-sequence into ED.

In either case, the message Edits will be lost – type Y to confirm: will appear to remind you that you will be replacing the current file.

Run File

Esc-RF

Loads and executes a command file of extended mode commands. The command file can consist of any ED commands, entered just as they would be within ED, on multiple lines if necessary. The file can be written and saved with any text editor. It can include function-key definitions, which may themselves be macros of several commands or complete editing processes, including exiting from ED. If no exit command terminates the command file, control will be returned to the keyboard at the end of the file's execution. ED will be in immediate mode.

**Undo****Esc-U**

Reverses changes made to the current line. Normally, when you move the cursor to a line, ED makes a copy of that line and stores the original. It is the copy that is changed when you add or delete characters. You can use Undo, while on the line, to bring back the original version. Once you move the cursor off the current line, ED makes the changed line a part of the file.

ED cannot undo a line deletion. Once you have moved from the current line, the U command cannot undo a change.

Show**Esc-SH**

Shows the current state of the editor. The screen displays information, such as the value of tab stops, current margins, block marks, and the name of the file being edited.

Set Tab**Esc-ST**

Sets the tab stop. To change the current setting of tabs, use the ST command followed by a number. For example:

ST 5

sets the tab stops to every fifth column.

Set Left Margin**Esc-SL**

Sets the left margin. To specify the left margin, use the SL command followed by a number indicating the column position. For example:

SL 10

sets the left margin to the 10th column. The left margin should not be set beyond the right edge of the screen. For instance, if you have an 80 column screen, you should not set the left margin to 81.

Set Right Margin**Esc-SR**

Sets the right margin. To specify the right margin, use the SR command followed by a number indicating the column position. For example:

```
SR 80
```

sets the right margin to the 80th column.

Extend Margins**Esc-EX**

Extends the margins for the current line. Once you have entered the EX command, ED ignores the right margin on the current line. When you move the cursor from the current line, ED turns the margins on again.

Status Line Message**Esc-SM**

Prints the given string on the status line. This is primarily useful when included in scripts that perform automated editing operations. It lets you display custom messages or prompt a user for input.

Save**Esc-SA**

Saves the text. If no filename is specified, SA saves to the current file. You can save to a different file via a file requester or by giving the name directly. For example, to bring up a file requester, type:

```
SA ? /Save as: /
```

This will display a file requester with **Save as:** in the title bar.

To save directly to a file, specify the name on the command line. For example:

```
SA !DF0:Doc/UpToDate!
```

saves the file as UpToDate, in the Doc directory on DF0:.

SA followed by Q is equivalent to the X command.

If slashes appear in the path to a file, do not use the slash as a delimiter.

Exit**Esc-X**

Exits ED saving the current file to the designated filename.

ED writes the text it is holding in memory to the file that was specified when ED was opened, then terminates. If you look at this file, you can see that all the changes you made are there.

If the T directory exists (usually in RAM:), ED also writes a temporary backup to SYS:T/Ed-backup. This backup file remains in the T directory until you exit from ED a second time. Then it is overwritten with the latest contents of ED. If the T directory does not exist, ED does not make a backup.

Exit with Query**Esc-XQ**

Exits ED unless changes have been made to the file. If changes have been made, a requester will ask if you really want to exit without saving the file. XQ is equivalent to clicking the Close gadget on the ED window.

Quit**Esc-Q**

Exits ED without saving changes. If you have made any changes to the file, ED will ask you if you really want to quit. If you press Y, ED terminates immediately without writing to the buffer and discards any changes you have made.

Cursor Control

There are several commands for moving the cursor around the screen. These are listed below:

Esc-T	Top of the file; first line of the file will be the first line on the screen
Esc-B	Bottom of the file; last line of the file will be the bottom line on the screen
Esc-EP	End of a page
Esc-PD	Next page
Esc-PU	Previous page

Esc-N	Start of next line
Esc-P	Start of previous line
Esc-CL	One place to the left
Esc-CR	One place to the right
Esc-CE	End of current line
Esc-CS	Start of current line
Esc-TB	Next tab position
Esc-WN	Start of next word
Esc-WP	Space after previous word
Esc-M <n>	To the line specified by <n>; for example:

M 503

moves the cursor to the 503rd line in the file.

Altering Text

The commands in this section describe ways to edit text on the screen.

Insert Before

Esc-I

Inserts the specified string on the line before the cursor.

Specify the string that you want to make into a new line after the I command, and the text will be inserted before the current line. For example:

I /Insert this before the current line/

inserts the string Insert this before the current line as a new separate line before the line containing the cursor.

Insert After

Esc-A

Inserts the specified string on the line after the cursor. This command works in the same way as I, except the string is inserted on a new line beneath the current cursor position.

Some other commands include:

Esc-S	Splits the current line at the cursor position
Esc-J	Joins the next line to the end of current line
Esc-D	Deletes the current line
Esc-DC	Deletes the character under the cursor
Esc-DL	Deletes the character to the left of the cursor
Esc-DW	Deletes to the end of the current word
Esc-EL	Deletes to the end of the current line
Esc-FC	Switches the case of letters

Block Control

To move, insert, or delete text, use the commands described in this section.

Block Start	Esc-BS
Block End	Esc-BE

Identifies the beginning and end of a block of text. To specify a block of text to be moved, inserted or deleted, move the cursor to the first line that you want in the block, and give the BS command. Then, move the cursor to the last line that you want in the block, and give the BE command.

NOTE: ED selects the entire line no matter where on the line the cursor is positioned.

If you have defined a block with BS and BE, then make changes to the text, the start and end of the block become undefined. You will have to redefine the block. The only exception to this is if you used the IB command to insert a block of text (explained below).

To specify one line as the current block, move the cursor to that line, press Esc, then type:

BS;BE

The current line becomes the current block. You cannot start or finish a block in the middle of a line. To do this, you must first split the line.

Insert Block**Esc-IB**

Inserts a copy of the block after the current line. The block will remain defined until you change the text. You can keep using IB to insert copies of the block throughout the document.

Delete Block**Esc-DB**

Deletes a block. Once you delete a block it becomes undefined. This means you *cannot* delete a block, then insert a copy of it. (You cannot use DB, then IB.) If you need to place a copy of it elsewhere, do that first, then delete it.

Show Block**Esc-SB**

Redraws the display so the block is at the top of the screen. The first line in the block will be at the top of the screen.

Write Block**Esc-WB**

Writes the block to another file. ED creates a file with the name that you specify, overwriting any other files with that name, then copies the block to the file. For example:

```
WB !DF0:Doc/Example!
```

writes the contents of the block to the Example file in the Doc directory. You can also specify the file through a file requester, by typing:

```
WB /File?/
```

Insert File**Esc-IF**

Inserts a file into the current file. ED reads into memory the specified file at the point immediately following the current line. For example:

```
IF !DF0:Doc/Example!
```

inserts the Example file into the current file. It is inserted on the line immediately after the line with the cursor in it.

Searching and Exchanging

ED allows you to search through the file for specific instances of text. You can substitute one pattern of text with another.

Find

Esc-F

Finds the next occurrence of the specified string of text. The search starts one character beyond the current cursor position and continues forward through the file. If the string is found, the cursor moves to the start of the located string. The string must be surrounded by acceptable delimiters (quotes, slashes, periods, or exclamation points). The search is case sensitive, unless the UC command is used (explained below).

For example:

`F !Workbench!`

searches through the file for the next occurrence of the word Workbench.

Backward Find

Esc-BF

Searches backwards through the file for the specified string. This command finds the last occurrence of the string before the current cursor position. The search continues through to the beginning of the file.

Exchange

Esc-E

Exchanges one occurrence of text with another. You must specify two strings, separated by delimiter characters. For example:

`E /SYS:/DF0:/`

changes the next occurrence of SYS: to DF0:.

ED starts searching for the first string at the current cursor position and continues through the file. After the exchange is completed, the cursor moves to the end of the exchanged text.

You can specify empty strings by typing two delimiters with nothing between them. If the first string is empty, ED inserts the second string at the current cursor position. If the second string is empty, ED searches for the next occurrence of the first string, then deletes it.

NOTE: ED ignores margin settings when exchanging text.

Exchange and Query

Esc-EQ

Searches for text to be exchanged, but asks for permission to do so. When you use EQ, ED asks you whether you want each exchange to take place. This is useful when you want the exchange to occur in some circumstances, but not in others. For example, after typing:

```
EQ /SYS:/DF0:/
```

ED finds SYS:, then displays an Exchange? message on the command line. If you press Y, the exchange takes place. If you press N, the cursor moves to the first place after the search string. EQ is usually given in repeated groups.

Upper/Lower Case

Esc-UC

Specifies a case-insensitive search. To tell all subsequent searches not to make any distinction between upper and lowercase text, use the UC command. To make searches case sensitive again, use the LC command.

Repeating Commands

ED remembers any extended command line you type. To execute a command line again, you do not have to retype it, you can press Ctrl-G. For instance, suppose you use Esc-F to search for a text string. If the first occurrence of the string is not the one you need, you can press Ctrl-G to execute the search command again. You can set up and execute complex sets of editing commands many times.

You can repeat a command a specified number of times, by entering the number before the command. For example:

```
4 E/rename/copy/
```

changes the next four occurrences of `rename` to `copy`.

You can use the RP (Repeat) extended command to repeat a command until ED returns an error, such as reaching the end of the file. For example:

```
T; RP E/rename/copy/
```

moves the cursor to the top of the file, then exchanges all occurrences of `rename` with `copy`. Notice that you need the T command (Top of File) to change all occurrences of `rename`. Otherwise, only the occurrences after the current cursor position would be changed.

To execute command groups repeatedly, you can group the commands together in parentheses. You can also nest command groups. For example:

```
RP (F /Workbench/; 3A//)
```

inserts 3 blank lines (the null string `//`) after every line containing `Workbench`. This command only works from the cursor to the end of the file. To apply the command to the entire file, you would have to first move to the top of the file.

To interrupt any sequence of extended commands, just press any key while the commands are being executed. If an error occurs, ED abandons the command sequence.

Customizing ED

This section describes the commands that relate to menu and function key setup. Remember, these commands can be entered individually within ED. They can also be saved as a script, such as S:Ed-startup, or as a file to be specified with the WITH argument. To execute the file from within ED, you could use the RF (Run Command File) extended command.

Set Menu Item

SI

Defines the menu headings and items. There are 120 menu item slots that you can fill. The syntax for SI is:

SI <slot number> <slot type> /string1/string2/

The slot numbers range from 0 to 119.

The slot type identifies the contents of the slot and is a number from 0 to 4. The possible types and their functions are listed below:

Type	Function	String Input
0	End of Menus	No arguments
1	Menu Heading	String1 = heading name
2	Menu Item	String1 = item name String2 = command string
3	Submenu Heading	String1 = heading name
4	Separator bar	No arguments

The 0 slot type must be the last defined slot. If you specify a menu heading, be sure to include menu items after it.

Do not create a menu without items. This could cause problems with your system.



Enable Menu

EM

Enables the menus. This command must be given after the SI commands in order for the commands to be enabled.

An example script is shown below. Quotes are used as the delimiters.

```
SI 0 1 "Project"
SI 1 2 "Open . . ." "op ? /Open file:/"
SI 2 2 "Save . . ." "sa"
SI 3 4
SI 4 2 "Quit!" "q"
SI 5 1 "Move"
SI 6 2 "Top" "t"
SI 7 2 "Bottom" "b"
SI 8 0
EM
```

This script would result in the following menu:



Set Function Key

SF

Defines the function keys. There are 57 immediate key slots. Defining function key and Ctrl-key commands is similar to defining menu items. The syntax is:

SF <slot number> /command string/

The slot numbers range from 1 to 57 with the function keys and Shifted function keys ranging from 1 to 20. Slots 21 to 25 are reserved for Shifted cursor key combinations and the Del key. The alphabetical Ctrl-key combinations range from 27 to 52 (one for each letter of the alphabet). To define Ctrl-key combinations, you can substitute a caret (^) and the other character for the slot number.

Many of the slots are already defined. For instance, slot 25, the Del key, is reserved for deleting the character at the cursor. Slot 27, Ctrl-A, inserts a new line after the current line. Any slot number may be redefined to execute any command string. Numbers within the range that do not appear are undefined.

An example script assigning function keys to cursor control commands is shown below. This could be combined with the menu script shown in the previous section. Quotes are used as delimiters.

```
SF 1  "t"  
SF 2  "b"  
SF 3  "ep"  
SF 4  "pd"  
SF 5  "n"  
SF 6  "p"
```

This script assigns the top of file, bottom of file, end of page, next page, next line, and previous line commands to the F1 through F6 keys, respectively.

A complete list of slots is shown on page 9-24.

Display Function Key

DF <key>

Displays the setting for the function key specified by <key>.
For instance, using the above example, if you pressed Esc-DF 1, the status line would display t to indicate that the T command is linked to F1.

Reset Key

RK

Resets the key definitions to the default.

Special Key Mappings		
Slot #	Key/Key Sequence	Function
1-10	F1 through F10	Not defined
11-20	Shift-F1 to Shift-F10	Not defined
21	Shift-left cursor	Move to beginning of line
22	Shift-right cursor	Move to end of line
23	Shift-up cursor	Move to previous page
24	Shift-down cursor	Move to next page
25	Del	Delete character at cursor
26	Not defined	Not defined
27	Ctrl-A	Insert line
28	Ctrl-B	Delete line
29	Ctrl-C	Not defined
30	Ctrl-D	Move down 12 lines
31	Ctrl-E	Move to top or bottom of screen
32	Ctrl-F	Change case
33	Ctrl-G	Repeat last extended command line
34	Ctrl-H	Delete character left of cursor
35	Ctrl-I	Move cursor to next tab position
36	Ctrl-J	Not defined
37	Ctrl-K	Not defined
38	Ctrl-L	Not defined
39	Ctrl-M	Return
40	Ctrl-N	Not defined
41	Ctrl-O	Delete word or spaces
42	Ctrl-P	Not defined
43	Ctrl-Q	Not defined
44	Ctrl-R	Move to end of previous word
45	Ctrl-S	Not defined
46	Ctrl-T	Move to start of next word
47	Ctrl-U	Move up 12 lines
48	Ctrl-V	Redisplay window
49	Ctrl-W	Not defined
50	Ctrl-X	Not defined
51	Ctrl-Y	Delete to end of line
52	Ctrl-Z	Not defined
53	Ctrl-[Esc (enter extended command mode)
54	Not defined	Not defined
55	Ctrl-]	Move to end or start of line
56	Not defined	Not defined
57	Not defined	Not defined

AREXX Support

ED may be controlled from AREXX. To do this, you need some familiarity with AREXX as well as three pieces of information: the name of ED's AREXX port, a method for sending commands to ED, and a way to retrieve information from ED.

Each running copy of ED has its own AREXX port name. If you are running multiple copies of ED, you must be careful to specify the correct port name for the corresponding ED session. For instance, the first session has the port name of ED, the second session is ED_1, the third session is ED_2, and so on.

Many of ED's extended commands can be used from AREXX. ED's RV command is used in AREXX programs to send information from ED to AREXX. It allows you to find out about the status of ED, such as the current line number or the name of the file being edited.

The RV command takes one argument—the name of the AREXX stem variable in which you would like the information stored. For example, the AREXX line:

```
address 'ED' 'RV /stem/'
```

assign values to the fifteen variables outlined below:

stem.LEFT	Current left margin (SL command)
stem.RIGHT	Current right margin (SR command)
stem.TABSTOP	Current tab stop setting (ST command)
stem.LMAX	Maximum visible line on screen
stem.WIDTH	Width of the screen in characters
stem.X	Physical X position on the screen (1 is the left edge)
stem.Y	Physical Y position on the screen (1 is the top line)
stem.BASE	Window base (normally 0, but non-zero when the screen is shifted to the right)

stem.EXTEND	Extended margin value (EX command)
stem.FORCECASE	Case sensitivity flag (UC/LC commands)
stem.LINE	Current line number in the file (1 is the first line)
stem.FILENAME	Name of the file being edited
stem.CURRENT	Text of the current line
stem.LASTCMD	Last extended command issued
stem.SEARCH	Last string searched for

NOTE: You can substitute any valid AREXX symbol for "stem." Be sure to enclose the name in proper delimiters, as shown above. These variables can then be treated as ordinary AREXX stem variables (see Chapter 10 for AREXX programming information).

You can use several extended commands from AREXX, as illustrated in the example program, transpose.ed, shown on page 9-27. This program, which you should launch from ED, will transpose two characters. For example, if a line contains the string 123 and the cursor is highlighting the 3, transpose.ed will change the string to 213.

To try this program, type it and save it as rexx:transpose.ed. Then, open ED and edit an existing file or just enter some new text. Place the cursor one character to the right of the two you want to transpose, press Esc, and type:

```
rx /transpose/
```

If you have typed the transpose.ed program correctly and AREXX is set up and running properly, the program will execute and the characters will be transposed.

Sample Program

```

/* Transpose.ed: An example program to transpose two characters. */
/* Given string '123', if cursor is on 3, this macro converts */
/* string into '213'. */

HOST = address ()      /* find out which ED session invoked this program */
address VALUE HOST     /* ...and talk to that session

'rv' '/CURR/'          /* Ask ED to store info in stem variable CURR

                        /* Obtain two pieces of information:
currpos = CURR.X        /* 1. position of cursor on line
currlin = CURR.CURRENT  /* 2. contents of current line

if (currpos > 2) then    /* Work only on the current line
    currpos = currpos - 1
else do                /* Otherwise, report error and exit
    'sm /Cursor must be at position 2 or further to the right/'
    exit 10
end

/* Next the code needs to reverse the CURRPOSTh and CURRPOSTh-1 characters
and then replace the current line with the new one.

drop CURR.            /* CURR is no longer needed; dropping it saves some memory.

'd'                   /* Tell ED to delete current line

currlin = swapch (currpos,currlin) /* Swap the two characters

'i '/'|currlin|'/'     /* Insert modified line
do i = 1 to currpos    /* Place cursor back where it started
    'cr'               /* ED's 'cursor right' command
end

exit                  /* Program has finished

/* Function to swap two characters */
swapch: procedure
parse arg cpos,clin
    ch1 = substr(clin,cpos,1) /* Get character
    clin = delstr(clin,cpos,1) /* Delete it from string
    clin = insert(ch1,clin,cpos-2,1) /* Insert to create transposition
return clin            /* Return modified string

```

MEmacs

MEmacs, which stands for MicroEmacs, is a screen-oriented editor that allows you to edit multiple files at one time. The only restriction is that the entire text file must be able to fit into memory at once, since MEmacs performs all of its operations on memory-resident text.

The length of the lines you can edit is limited to the right-hand edge of the screen, usually 80 characters. Characters beyond the rightmost character of the line are not lost; they simply do not show on the screen. The only way to see those characters is to break the line or to delete some of the displayed characters. When entering new characters, you can keep typing past the rightmost character on the line, but what you type will not show on the screen.

Starting MEmacs

MEmacs is in the Tools directory of the Extras2.0 disk. You can run MEmacs from either the Workbench or the Shell. From the Workbench, double-click on the MEmacs icon in the Tools window of the Extras2.0 disk. If you have a hard disk, the Tools drawer will be in your System2.0 window.

From the Shell, the format is:

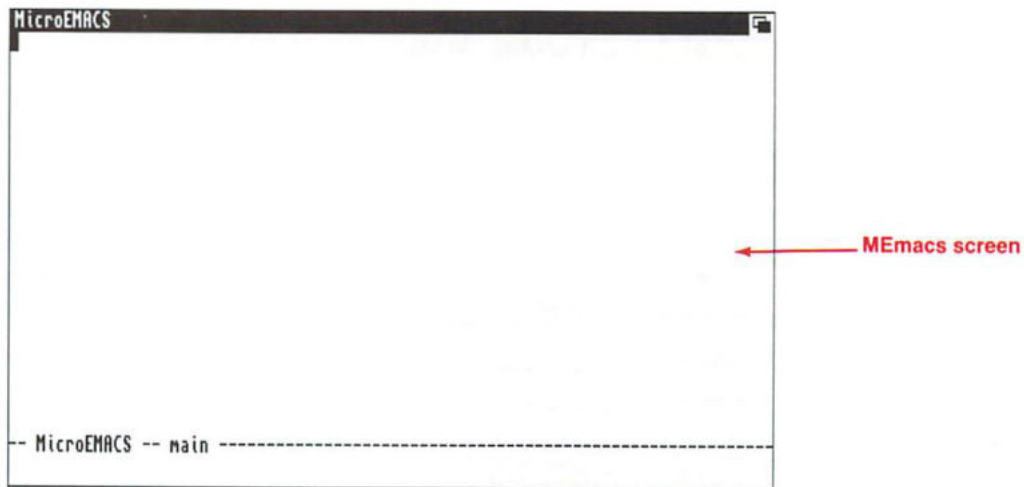
Format: MEmacs [<filename>] [goto <n>] [OPT W]

<Filename> specifies the file to read into MEmacs. If no file of that name can be found, the file will be created when you save your work. This argument is optional. The goto <n> option allows you to specify the line that the cursor will appear on when the file is opened. This is convenient if you are editing a particularly large file.

Normally, MEmacs opens on a new screen. However, if you specify the OPT W option, MEmacs opens in a Workbench window.

Using MEmacs

When you open MEmacs, without specifying a filename, the words MicroEMACS — main appear at the bottom of the screen.



This line displays the name of the buffer that is currently in use. A buffer is an area of memory that MEmacs is using to store the text that you are editing.

If you had specified a filename, the file will be read into the buffer, and MEmacs will give the buffer the same name as the file. In that case, the bottom line of the screen will display the name of the buffer along with the filename with which it is associated.

You can have several buffers in use at one time, and you can see one or more on the screen at the same time. Menu options let you switch back and forth between them. At all times, what you see on the screen is what is actually in the buffer.

MEEmacs has two modes of operations: normal and command. When MEEmacs is in normal mode, you can:

- move the cursor using the cursor keys
- move the cursor to the edge of the window by holding down Shift and pressing the appropriate cursor key
- move the cursor by clicking the left mouse button at the desired place on the screen
- insert characters at the current cursor position simply by typing them
- delete the character at the current cursor position by pressing Del
- delete the character to the left of the cursor by pressing Backspace
- perform other special functions as explained in the menu section and command summaries that follow

When MEEmacs is in the command mode, the cursor jumps to the bottom line of the display, and the program asks you for certain additional information. The command mode is entered through various menu items which are explained later.

There are some special terms associated with MEEmacs that you should be familiar with:

Buffer	A memory area that MEEmacs controls. There is always at least one buffer used by MEEmacs, and it can contain zero or more characters of text.
Dot	The current cursor position.

Mark A cursor position that you have specified. (Each buffer has its own dot and mark.) The Set-mark menu item allows you to mark the current cursor position. You can then move forward or backward in the file, adding or deleting text. Then, when you wish to return to the place that you marked, you simply select the Swap-dot&mark menu item.

You can also set a mark to indicate the beginning of a block of text that you want to duplicate, move, or delete. The block will encompass all the characters starting with the mark and continuing to the current cursor position.

Kill Kill commands remove text from the screen and save it in a kill buffer. This text can be retrieved and put back into your document by using the Yank command. As you issue successive Kill commands (without selecting Yank in between), each block of text that you kill will be *added* to the existing text in the kill buffer.

Window A MEmacs window is somewhat different from Workbench window. In MEmacs, the screen can be split into multiple layers so that you can edit and display more than one buffer, or two or more portions of the same buffer. Each layer is an MEmacs window.

**Modified
Buffers**

When you make any changes to a buffer, even if you only press Return then delete it, MEmacs remembers and will mark that buffer as a modified buffer.

You can see which buffers have been modified by using the List-buffers command. Any modified buffers are identified with an asterisk (*). If you try to exit MEmacs without saving any changes, a prompt will tell you that modified buffers exist and will ask you if you really want to quit. Once you save a buffer, the modified status is removed.

Menu Commands

MEmacs has the following menus:

Project	Contains system and file-oriented items.
Edit	Contains buffer editing commands.
Window	Controls the characteristics of the MEmacs windows.
Move	Controls the placement of the cursor.
Line	Controls line-oriented operations.
Word	Controls word-oriented operations.
Search	Controls search and search/replace options.
Extras	Controls the numerical value of arguments, and lets you execute a series of operations as though it were a single special command.

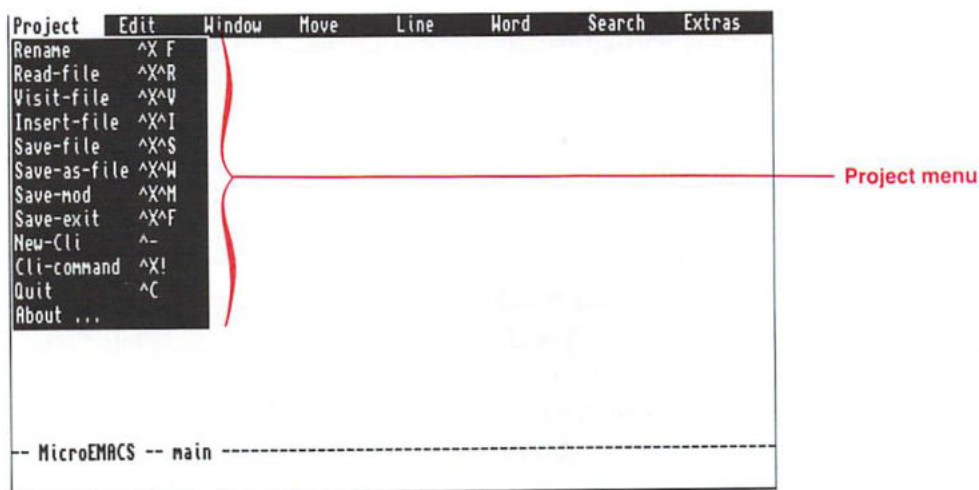
This section explains each of these menus and their related commands. Each of the commands also has a keyboard

shortcut. The shortcuts appear to the right of the menu item. In the menu, Ctrl is represented by a caret (^). For instance, the Rename menu item shows ^X F as its shortcut. You must press Ctrl-X-F.

In the manual, the keyboard shortcuts are shown along the right margin and the standard format for key sequences is used. However, if a symbol is shown, you must press Shift to create that symbol. For instance, the keyboard shortcut for the Set-Mark menu item is Ctrl-@. Since the @ symbol is created by pressing Shift-2, you must press Ctrl-Shift-2; Ctrl-2 will not work.

The Project Menu

The commands in the Project menu, except for Visit-file, affect the buffer associated with the current cursor position.



Rename

Ctrl-X-F

Changes the name of the file associated with the current buffer. This command is useful if you are saving versions of a program or text file as you go along. You can perform a Save

command for the first version, modify a few things, rename the file associated with this buffer and then save the new version.

When you select Rename, MEmacs prompts:

New file name:

If you press Return without specifying a filename, the buffer becomes disassociated from any filename. You must specify a name here if you want the buffer to be appropriately associated with a file.

Read-file

Ctrl-X-Ctrl-R

Replaces the contents of the current buffer with the contents of a file. When you select Read-file, MEmacs moves the cursor to the bottom line of the display and prompts:

Read file:

Enter the complete path to the file, including the volume name, directory, and file, then press Return. The file is read into the current buffer, overwriting the data that was stored there.

If you do not want to read a file, simply press Return without specifying a filename. MEmacs will ignore the request and return you to normal mode.

Visit-file

Ctrl-X-Ctrl-V

Lets you work with additional files, aside from the first file you open. You must already be editing something before you can visit another file. This command is useful for programmers who are creating a program and want to extract pieces from or refer to other programs.

When you issue this command, MEmacs moves the cursor to the bottom line and prompts:

Visit file:

Type the complete path of the file, and press Return. MEmacs will read the file into a buffer, if it is not already there. If the file you want to visit is on a different disk, AmigaDOS will display a requester asking you to insert that particular disk into any drive. If the file is already in a buffer, MEmacs will switch you to that buffer automatically.

Insert-file**Ctrl-X-Ctrl-I**

Inserts the contents of a file into the current buffer. When you issue this command, MEmacs moves the cursor to the bottom line and prompts:

Insert file:

Enter the complete path to the file, and press Return. MEmacs will read it into the current buffer at a point one line above the current cursor position.

Save-file**Ctrl-X-Ctrl-S**

Writes the contents of the current buffer to the filename associated with that buffer. The filename associated with the buffer was determined when the contents of an existing file were read to the file or when the file associated with the current buffer was renamed.

If there is no filename specified on the status line, MEmacs tells you No File Name and refuses to perform the Save.

After a successful Save, MEmacs uses the bottom line of the screen to tell you how many lines it has written out to the designated file.

Save-as-file**Ctrl-X-Ctrl-W**

Allows you to specify the name of a file to associate with a buffer. When you issue this command, MEmacs prompts:

Write file:

MEmacs is requesting the name of the file in which it should save the current contents of the buffer. If you provide a

complete path and press Return, the buffer will be written out to that disk, directory, and filename. (If you press Return without providing a name, you are returned to normal mode.) The following notation appears on the status line:

File: <filename>

From now on, that file will be used to save the current contents of this buffer when you issue a Save command.

Save-mod**Ctrl-X-Ctrl-M**

Writes the contents of all modified buffers to the disk. Use this item with caution to ensure that you don't accidentally modify a buffer associated with a file you have visited but don't intend to change.

Save-exit**Ctrl-X-Ctrl-F**

Saves all modified buffers then exits MEmacs. It is simply a combination of the Save and Quit items. Again, use this item with caution (see the menu item Save-mod).

New-Cli**Ctrl- -**

Brings up a new Shell window called Spawn Window. You can issue as many AmigaDOS commands in the spawn window as you want without interfering with MEmacs. To return to MEmacs, use the ENDSHELL command. The spawn window disappears, and MEmacs is restored to its previous state.

Cli-Command**Ctrl-X-!**

Allows you to execute an AmigaDOS command while you are still in MEmacs. It is similar to issuing a RUN command while in the Shell.

When you select this menu item, MEmacs moves the cursor to the bottom of the screen and provides you with a ! prompt. You can then type a command for AmigaDOS to process on this line. MEmacs temporarily suspends operation, and AmigaDOS executes your command. The output of the command appears in a temporary buffer called spawn.output.

Quit**Ctrl-C**

Exits MEmacs. If one or more of the buffers has been modified since you last saved it to a file, MEmacs prompts:

Modified buffers exist, do you really want to exit? [y/n]?

MEmacs is giving you a last chance to save your work. If you don't want to exit, simply press Return. If you do want to quit, press Y then press Return.

Before quitting, you can check existing buffers by selecting List-buffers in the Edit menu. MEmacs lists the names associated with each buffer and shows an asterisk by each buffer that has been modified since you last saved it to disk.

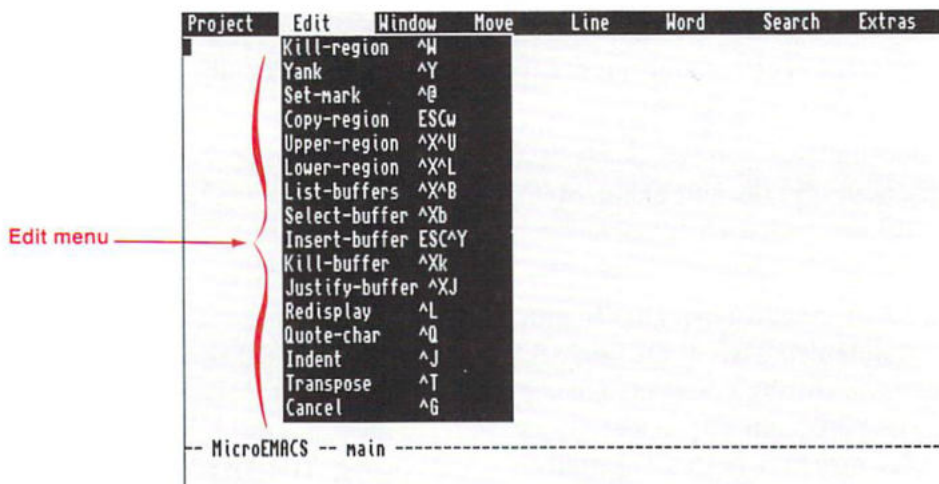
There are circumstances under which you will not want to save all buffers back to the original files. For example, suppose you were writing a program and copying pieces from other existing programs as you went along. Some of the files you visited may have been accidentally modified or may have been on a write-protected disk.

If you are simply using an old program as temporary source material, you will not want to destroy the original program. When you are finished writing the new program, save your new material and exit MEmacs without saving the modified buffers of the source program.

Two alternative keyboard shortcuts for the Quit command are Ctrl-X-Ctrl-C and Esc-Ctrl-C.

The Edit Menu

The commands in the Edit menu affect the editing of your buffers and their associated files.



Kill-region

Ctrl-W

Deletes blocks of text from the current buffer and saves them in a kill buffer. (Text can be pulled back into the document by using the Yank command, described below.)

If a block of text has been marked using the Set-mark command (explained below) and the cursor has been positioned away from the mark, the area between those two points is considered a block and can be deleted by selecting Kill-region.

You can also use Kill-region to copy a block from one section of the buffer to another. Mark the block, select Kill-region, then *without moving the cursor*, immediately select Yank. The block will be restored to its original position, but there will also be a copy of the block in the kill buffer.

If you repeatedly select Kill-region on different areas of text, without performing a Yank, each successive kill segment is

appended to the kill buffer. When you perform the first Yank, it marks the end of the kill buffer.

Yank**Ctrl-Y**

Copies the contents of the kill buffer to the line immediately above the current cursor location in the current buffer. Yank reverses the action of Kill-region, but it does not change the contents of the kill buffer. Therefore, you can repeatedly move the cursor to another buffer, select Yank, and copy the contents of the kill buffer. The next time you kill a block of text, however, the contents of the kill buffer will be replaced with the new material and the old contents will be lost.

Kill-region and Yank are often used together to move text from one buffer to another.

Set-mark**Ctrl-@**

Marks the cursor position in a buffer. When you select Set-mark, the position of the cursor is marked in the current buffer. From then on, any other position of the cursor is referred to as the dot. You can move back and forth between the mark and the dot by selecting the Swap-dot&mark command in the Move menu.

You can use Set-mark to mark the beginning of a block of text that you want to duplicate or move somewhere else in the buffer. Set the mark on the first character you want to include in the block. As you move the cursor through the file, you are essentially blocking out a portion of text.

An alternative keyboard shortcut for Set Mark is Esc - -.

Copy-region**Esc-W**

Copies the contents of the marked region to the kill buffer. This new text replaces any previous contents of the kill buffer.

Upper-region**Ctrl-X-Ctrl-U**

Changes the text of the entire marked region, the area between the mark and the current cursor position (dot), to uppercase.

Lower-region**Ctrl-X-Ctrl-L**

Changes the text of the entire marked region to lowercase.

List-buffers**Ctrl-X-Ctrl-B**

Splits the current buffer's window and provides you with a list of the buffers that MEmacs is currently maintaining. The list has 4 columns. For example:

C	Size	Buffer	File
*	17260	Emacs.doc	df1:Docfiles/Emacs.doc

The fields are:

C	Displays an asterisk if the buffer has been modified since it was last saved to a file. (Stands for "changed.")
Size	Shows how many characters are in the buffer.
Buffer	Shows the name given to the buffer. If you have read in a file, this will usually be the name of the file itself, minus the full path. In the example above, the file being edited is DF1:Docfiles/Emacs.docs, but the buffer name is just Emacs.docs.
File	Shows the full path to the file. This is the file where MEmacs will write the buffer if you choose Save-file or Save-exit while the cursor is in that buffer.

When you choose List-buffers, the status line at the bottom of the screen displays MEmacs — [List]. Even though List-buffers brings up a window display, it is not listed as an available buffer. If you edit the List-buffers window, it can be made to act just like any other buffer. If, for example, you open a file in the List-buffers window, the name of the buffer will continue to be [List], and the name of the file you have opened will become associated with the List-buffers window.

If you should leave the List-buffers window on the screen but use a different window to modify the listed buffers, the List-buffers display will not be continuously changed to reflect the current changes. To get current information, you must select List-buffers again.

Select-buffer

Ctrl-X-B

Lets you select which buffer you wish to edit in the currently selected window, the window where your cursor is positioned. When you choose Select-buffer, MEmacs moves the cursor to the bottom line and prompts:

Use buffer:

You must provide a name that is the same as one of those shown in the List-buffers listing. If you specify one of the available names, that buffer replaces the contents of the currently selected window.

If you specify a name that is not in the List-buffers listing, you are telling MEmacs to create a new buffer with that name. In this case, there is no filename associated with the new buffer and you will have to rename the file or select Save-as-file when you are prepared to save the buffer's contents to a file.

If you simply press Return, the command is ignored.

Insert-buffer**Esc-Ctrl-Y**

Inserts the contents of a named buffer into the current buffer at the line above the current cursor position. When you select Insert-buffer, MEmacs prompts:

Insert buffer:

You must type the name of the buffer to insert, then press Return.

Kill-buffer**Ctrl-X-K**

Deletes the contents of a chosen buffer. MEmacs can only edit a file if the entire file will fit in available memory. To make room in memory, you can use Kill-buffer to delete the contents of one or more buffers. This command returns the buffer's memory to the memory manager for reuse.

When you choose Kill-buffer, MEmacs prompts:

Buffer to kill (delete):

You must then enter the name of the buffer you wish to delete. You cannot kill a buffer if its contents are currently displayed.

Justify-buffer**Ctrl-X-J**

Removes all blank spaces and tabs from the left-hand edge of all the lines in the current buffer. The text is rearranged so that it aligns with the current margins.

Redisplay**Ctrl-L**

Redraws the screen.

Quote-char**Ctrl-Q**

Allows you to insert a literal character in the text file. Some keyboard selections have been assigned as MEmacs control characters (for instance, the menu command shortcuts). If you try to insert such a selection into your text, MEmacs will react as if you chose a menu item.

For example, Ctrl-L tells MEmacs to redraw the display, but Ctrl-L is also useful as a printing control to insert a form feed character. By selecting Quote-char, the next character you type will be taken “literally” by MEmacs and will be inserted into the text file, instead of being treated as a menu command.

To quote the form feed character, press Ctrl-Q-Ctrl-L. MEmacs will display `^L`, on the screen. (MEmacs uses the caret (`^`) symbol to represent Ctrl.)

As MEmacs manipulates the buffer, the combination of the caret and the character is treated as a single character, both by the cursor keys and the character counter.

You can also use Quote-char to insert a Return (Ctrl-M), Backspace (Ctrl-H), or Esc (Ctrl-[]) into the text by quoting the single keys, or for inserting any other control character that may be needed during a macro command. Even Ctrl-Q can be inserted by typing it twice. The Tab key cannot be quoted.

An alternative keyboard shortcut for Quote-char is Ctrl-X-Q.

Indent **Ctrl-J**

Moves the cursor to the next line, automatically indenting the same amount of spaces as the previous line.

Alternative keyboard commands are Help or Enter on the numeric keypad.

Transpose **Ctrl-T**

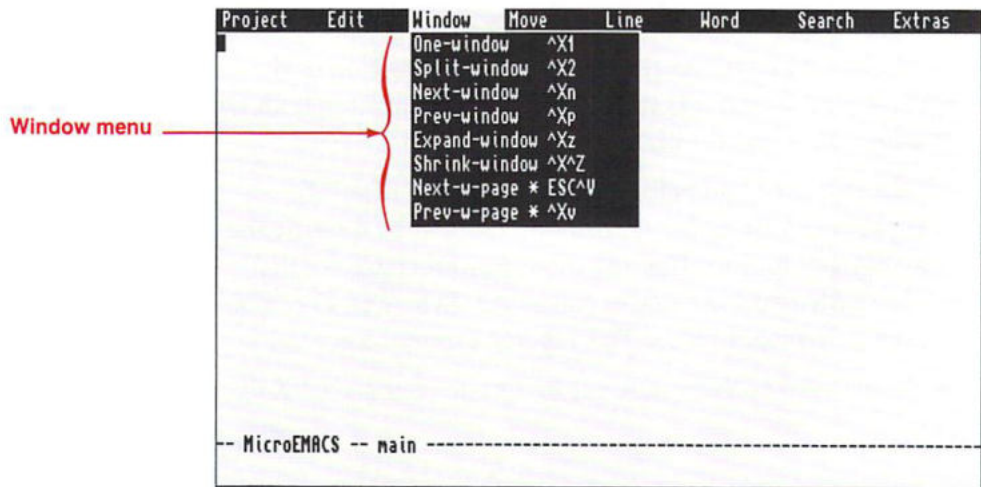
Swaps the positions of two adjacent characters. Place the cursor on the right-most of the two characters.

Cancel **Ctrl-G**

Ends an ongoing menu command, such as a query search and replace.

The Window Menu

A window in MEmacs is not the same as a window on the Workbench. MEmacs splits the screen into multiple layers, allowing you to edit a separate file (buffer) in each MEmacs window. The Window menu lets you control how you view your buffers on the screen.



One-window

Ctrl-X-1

Makes the current buffer a single, full-sized window on the MEmacs screen. All other buffers remain invisible, allowing you maximum space to work on the current buffer.

Split-window

Ctrl-X-2

Splits the current window in half, positioning the current buffer identically in both windows. This lets you edit two segments of the buffer at the same time. Any changes made in either window affect the entire buffer. This is convenient when you want to see what you wrote in an earlier part of your document while working on a later section.

Next-window**Ctrl-X-N**

Moves the cursor to the next window and makes that window available for editing.

If the cursor has been moved down to the bottom window, the cursor will automatically move up to the top window.

Prev-window**Ctrl-X-P**

Moves the cursor to the previous window and makes that window available for editing.

Selecting Prev-window when the cursor is in the top window will move the cursor to the last, or bottom, window.

Expand-window**Ctrl-X-Z**

Adds a line to the height of the current window and simultaneously subtracts a line from the height of the adjacent window.

Shrink-window**Ctrl-X-Ctrl-Z**

Subtracts a line from the height of the current window and simultaneously adds a line to the height of the adjacent window.

Next-w-page**Esc-Ctrl-V**

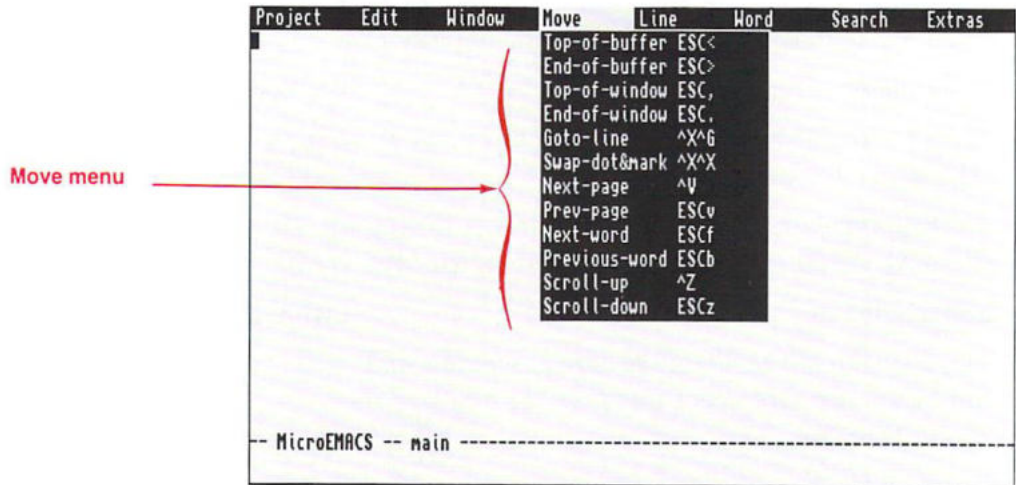
Displays the next page of the adjacent window. For instance, if you have split a window and are working in the top one, selecting Next-w-page will move the contents of the bottom window (the one you aren't working in) to the next page. This doesn't make the window available for editing; it just lets you view the contents.

Prev-w-page**Ctrl-X-V**

Displays the previous page of the adjacent window. If only one window is displayed, it displays the previous page of that window.

The Move Menu

The commands in the Move menu let you move the cursor rapidly through the current buffer.



Top-of-buffer

Esc-<

Moves the cursor to the top line of the current buffer.

End-of-buffer

Esc->

Moves the cursor to the bottom line of the current buffer.

Top-of-window

Esc-,

Moves the cursor to the top of the current window.

End-of-window

Esc.,

Moves the cursor to the bottom of the current window.

Goto-line

Ctrl-X-Ctrl-G

Moves the cursor to a specific line number. When you select Goto-line, MEmacs moves the cursor to the bottom of the screen and prompts:

goto-line:

Enter a line number, press Return, and MEmacs moves the cursor directly to that line. If you specify a line number larger than the total number of lines in the buffer, MEmacs moves the cursor to the last line of the buffer.

Swap-dot&mark**Ctrl-X-Ctrl-X**

Places a mark at the current cursor position and moves the cursor to where the mark had been set. If you have not yet set a mark in the window, MEmacs replies, No mark in this window. This command lets you move quickly to and from a preset location in your buffer. Selecting this item again restores the cursor to where it was before you selected Swap-dot&mark the first time.

Next-page**Ctrl-V**

Moves the text within the window toward the end of the buffer by one full window, less one line. The cursor is repositioned so as to stay on the screen.

Prev-page**Esc-V**

Moves the text within the window toward the beginning of the buffer by one full window, less one line. The cursor is repositioned so as to stay on the screen.

Next-word**Esc-F**

Moves the cursor forward to the next non-alphanumeric character, such as a space or punctuation mark, after the current word.

Previous-word**Esc-B**

Moves the cursor back to the first letter of the previous word.

Scroll-up**Ctrl-Z**

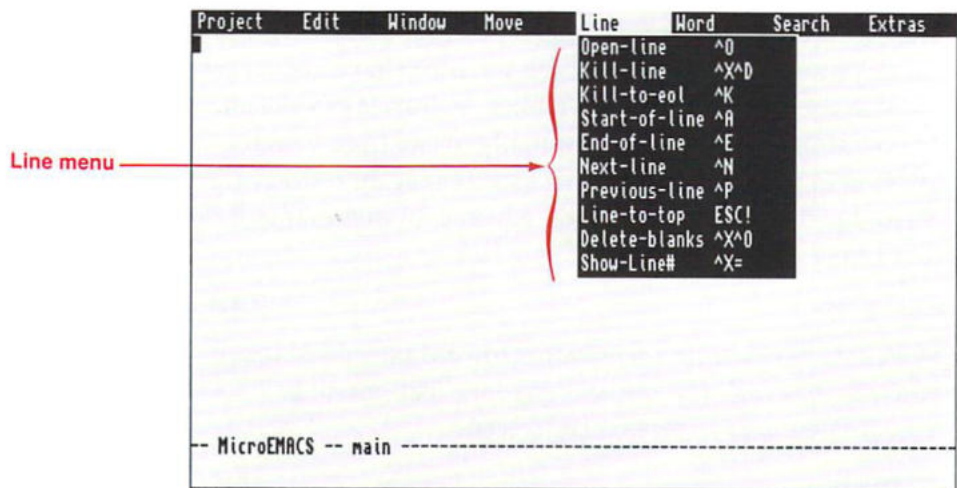
Moves the text up a single line.

Scroll-down**Esc-Z**

Moves the text down a single line.

The Line Menu

The commands in the Line menu let you move the cursor within or between lines and let you perform operations involving entire lines.



Open-line

Ctrl-O

Splits the line the cursor is in, forcing the character on which the cursor rests to become the first character of the following line. This command leaves the cursor in the original line so that you can type new characters beginning at the current cursor position.

If you select Open-line by mistake, immediately pressing Del closes up the line.

Kill-line

Ctrl-X-Ctrl-D

Deletes the line in which the cursor is located and places the text in the kill buffer. If you have not selected Yank since the last Kill command, the text will be appended to the existing text in the kill buffer.

Kill-to-eol**Ctrl-K**

Deletes the text between the current cursor position and the end of the line. If you have not selected Yank since the last Kill command, the text will be appended to the existing text in the kill buffer.

Start-of-line**Ctrl-A**

Moves the cursor to the left-most position on a line.

End-of-line**Ctrl-E**

Moves the cursor to the right-most position on a line. If you have typed more characters than will fit on a line, a dollar sign (\$) appears at the right-hand edge of the line. Moving to the end of the line places the cursor logically on the right-most character even though you cannot see it. Physically the cursor is positioned over the dollar sign. If you use the left cursor key to move the cursor, it will take as many key presses as there are unseen characters before the cursor actually begins to move.

NOTE: If you invoke MEmacs with OPT W to use a Workbench window for MEmacs, and your display mode allows more than 80 columns, MEmacs displays as many characters as will fit before showing the dollar sign.

Next-line**Ctrl-N**

Moves the cursor down one line.

Previous-line**Ctrl-P**

Moves the cursor up one line.

Line-to-top**Esc-!**

Moves the line containing the cursor to the top of the window.

Delete-blanks**Ctrl-X-Ctrl-O**

Deletes blank lines, proceeding forward from the current cursor position until MEmacs gets to the next line on which text exists.

Show-Line#**Ctrl-X-=**

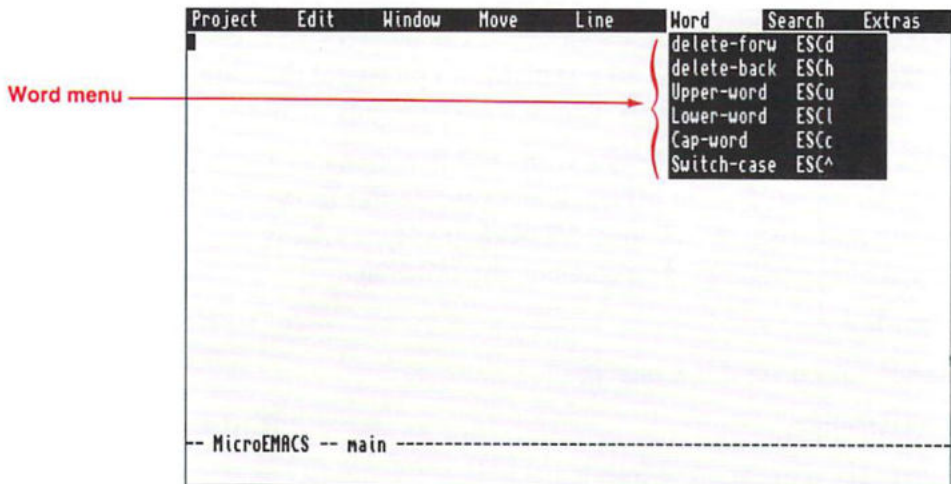
Displays information on the present cursor position. For example:

Line 17 Column 1 (2%)

In this example, the cursor is on the 17th line of text, in the first column. The percentage shows that the cursor is in a position 2% of the way from the top of the buffer. If the cursor was on the last character of text, the percentage would be equal to 100.

The Word Menu

The Word menu contains word-associated operations.

**Delete-forw****Esc-D**

Deletes the character on which the cursor is positioned and all remaining characters to the right until the next non-alphanumeric character is found, (i.e., a blank space, tab, or punctuation mark).

For instance, if the cursor is positioned on the "s" in the word "wordsuffix," choosing Delete-forw will delete "suffix" from the word. If the cursor is positioned on a blank space, it must be moved forward to the start of a word to delete that word.

Delete-back**Esc-H**

Deletes all characters to the left of the cursor until it finds the first character of a word. The character under the cursor is not deleted.

An alternative keyboard shortcut for this command is Esc-Del.

Upper-word**Esc-U**

Changes a word to uppercase, starting at the character where the cursor is positioned and proceeding to the last character of the word.

Lower-word**Esc-L**

Changes a word to lowercase, starting at the character where the cursor is positioned and proceeding to the last character of the word.

Cap-word**Esc-C**

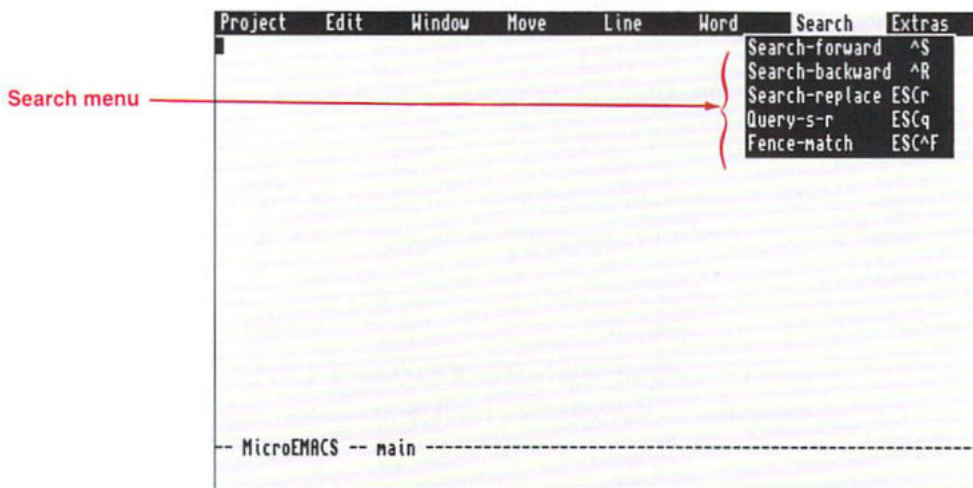
Changes the character where the cursor is positioned to uppercase. It also changes the characters to the right of the cursor, up to the end of the word, to lowercase.

Switch-case**Esc-^**

Changes the case of a word, starting at the current cursor position and proceeding to the right until it reaches the end of the word. If a word is uppercase it changes it to lowercase, and vice versa.

The Search Menu

The Search menu allows you to search through the current buffer for specific text strings. The case (upper or lower) of the string is not significant in the search itself. However, if you are using text substitution (search and replace), the text will be replaced in the same case as that of the replacement string.



Search-forward

Ctrl-S

Searches through the text starting at the current cursor position and moving forward to the end of the buffer. When you issue this command, MEmacs moves the cursor to the bottom line of the screen and prompts:

Search:

Enter the string of characters that you want MEmacs to search for, and press Return. If the string is found, MEmacs positions the cursor immediately following the last character of the string.

If MEmacs cannot find the string, it replies Not found.

An alternative keyboard shortcut for this command is Ctrl-X-S.

Search-backward**Ctrl-R**

Searches through the text from the current cursor position backwards to the beginning of the buffer. This command operates in the same manner as Search-forward.

An alternative keyboard shortcut for this command is Ctrl-X-R.

Search-replace**Esc-R**

Operates the same way as Search-forward, except that it allows you to replace the string with different text. When MEmacs finds the first occurrence of a specified string, it prompts:

Replace:

You must enter the string of characters that should replace the found string. Remember, the characters will appear in the same case as you type them. When you press Return, MEmacs will automatically forward-search and replace the search string with the replacement string. After MEmacs completes this command, it reports:

Replaced <xx> occurrences

<xx> stands for the number of times the string was replaced.

Query-s-r**Esc-Q**

Operates the same way as Search-replace, except that it allows you to choose whether or not to replace each occurrence of the string. When you select Query-s-r, MEmacs prompts for the search string, then prompts:

Query replace:

As it finds a matching string, it prompts:

Change string? [y/n/c/^G]?

The options are: Y (yes); N (no); C (change all occurrences of the string); and Ctrl-G (abort). This gives you a chance to control the replacement process. After MEmacs completes this command, it reports:

Replaced <xx> occurrences

Fence-match**Esc-Ctrl-F**

Finds the closest occurrence of the fence character to match the one at the current cursor position. A fence character is the closing character to match a:

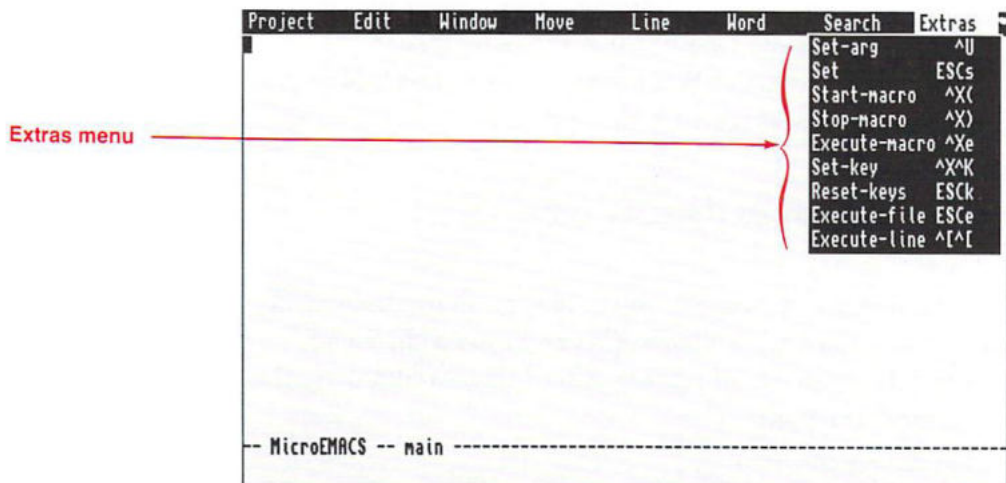
parenthesis	(matches)
bracket	[matches]
brace	{ matches }
angle bracket	< matches >

If you choose Fence-match while the cursor is on an opening parenthesis, the cursor will move to the next occurrence of a closing parenthesis.

If you choose Fence-match while the cursor is on another type of character, such as a letter or symbol, the cursor will move to the next character of the same type. For instance, an asterisk matches another asterisk.

The Extras Menu

The Extras menu contains commands to let you tell MEmacs how to operate. Many of these operational commands require that you specify a numeric argument before selecting the command itself.



This menu also includes several **macro** commands. A macro command is actually a sequence of commands or other keystrokes that are executed by selecting the Execute-macro menu item.

Set-arg

Ctrl-U

Lets you specify a numeric argument for a command. When you issue this command, MEmacs prompts:

Arg: 4_

If you select Set-arg again, MEmacs multiplies the argument value by 4.

If you press a numeric key (0-9), MEmacs accepts an integer argument. If you press a minus sign first, MEmacs accepts a negative integer argument, starting at -1.

Examples: (Each started by a single press of Ctrl-U)

Arg: -1	Pressed - as the first key
Arg: -23	Pressed - - 2 - 3 as a 3-key sequence

MEmacs accepts the argument value as a key for whatever you do next. To add 12 blank lines at the cursor position, specify an argument of 12, then press Return. To add 20 minus signs, select an argument number of 20, *do not press Return*, and press the minus sign on the keyboard.

NOTE: Don't use the keypad's minus sign; it is mapped to a different value.

To set one of the MEmacs operational parameters (described below), select the value of the argument, *do not press Return*, then select the appropriate menu item. MEmacs will use the argument to set the value.

If the command does not support parameters. MEmacs executes the command the specified number of times.

Set**Esc-S**

Allows you to choose various MEmacs parameters. When you choose Set, MEmacs prompts:

Set:

You can then enter one of the following:

- | | |
|-----------|--|
| Screen | Places the MEmacs display in a Workbench window or back onto a custom screen. |
| Interlace | Turns the interlace mode on or off. |
| Mode | Results in a second prompt Mode; you can enter cmode (for editing c programs) or wrap (to enable automatic word-wrap when the text reaches a set cursor position). Cmode provides automatic fence matching. Use + mode or - mode to add or subtract a mode. |
| Left* | Determines the left margin. |
| Right* | Determines the right margin. |
| Tab* | Sets the increment for tab spacing. |
| Indent* | Determines how far to indent each level of nesting (used in c mode). |
| Case | Turns case sensitive searches on or off; default is off. |
| Backup | Turns the MEmacs backup function on or off. Your options are: ON (renames the current file <filename>.bak and saves that backup file to the T: directory); SAFE (this option checks to see if a file already exists for the buffer—if so, it will not overwrite the existing file); and OFF (this is the default option—MEmacs does not perform any backup). |

**Each of these entries results in a prompt for a numerical argument, unless the numeric argument is given along with the entry.*

Start-macro**Ctrl-X-(**

Tells MEmacs to start recording any subsequent keystrokes.

This is a macro command and is used in conjunction with the Stop-macro and Execute-macro commands.

Stop-macro**Ctrl-X-)**

Tells MEmacs to stop recording keystrokes.

Execute-macro**Ctrl-X-E**

Repeats keystrokes and menu selections that were entered between Start-macro and Stop-macro. They are repeated as if you had freshly entered the entire sequence.

Set-key**Ctrl-X-Ctrl-K**

Allows you to redefine all of the function keys, the Shifted function keys, the Help key, or any key on the numeric keypad as keyboard macros. This means that if you select one of these redefined keys while recording macro commands, the new key definition will be recorded in the command. One definition, having as many as 80 keystrokes, can be recorded for each of these keys.

NOTE: If you want to insert the Set-mark command into any of the keyboard macro definitions, you can't use the menu shortcut of Ctrl-@. This does not function correctly when used in a macro command. Instead, you must use the alternative form of Set-mark, Esc- -. This alternative form is acceptable in macro commands.

When you choose Set-key, MEmacs prompts:

key to define:

Press one of the 10 function keys, Help, or a numeric keypad key. MEmacs responds:

def: [commands]:

[commands] is a display of the current commands bound to that key. Enter the new string of characters (up to 80) that you

want to have MEmacs respond to when this key is touched. Pressing Return terminates the entry.

Remember that when entering commands that involve function keys, for example Esc-< (go to top of buffer), you must use Quote-char (Ctrl-Q) to properly insert the keystroke into the definition.

The table below contains the default values of the function keys when used in macro commands.

Default Function Keys Assignments		
Key	Assignment	Key Sequence
F1	Clone line	Ctrl-A-Ctrl-K- Ctrl-Y-Ctrl-M- Ctrl-Y
F2	Delete line	Ctrl-X-Ctrl-D
F3	Execute keyboard macro	Ctrl-X-E
F4	Next screen	Ctrl-V
F5	Previous screen	Esc-V
F6	Split window	Ctrl-X-2
F7	One window	Ctrl-X-1
F8	Scroll window up	Ctrl-Z
F9	Scroll window down	Esc-Z
F10	Save file and exit	Ctrl-X-Ctrl-F
Help	Insert line	Ctrl-J
Enter (keypad)	Insert line	Ctrl-J

The numeric, period, and minus keys on the numeric keypad default to their normal values (i.e. keypad 1 defaults to 1, keypad 2 defaults to 2, etc.).

Reset-keys

Esc-K

Returns any keys defined by Set-keys to their original default state.

Execute-file**Esc-E**

Allows you to execute a program file within MEmacs. When you select this command, MEmacs prompts:

File:

Enter the name of the file you wish to access. This file is executed as a file of MEmacs commands.

Execute-line**Ctrl-[-Ctrl-]**

Sets MEmacs to the command mode. When you choose Execute-line, MEmacs prompts:

execute-line:

You can then enter any menu command and its parameters by simply typing it at the prompt. You must use the exact format used in the menus, including hyphens, or you will receive an alert and command error message. For instance, this is *incorrect*:

execute-line: insert file <filename>

You must type:

execute-line: insert-file <filename>

An alternative keyboard shortcut for Execute-line is Esc-Esc.

Commands Not in Menus

The following commands have not been installed in menus and are only accessible through the keyboard.

Describe Key**Esc-Ctrl-D**

Tells you if any functions are bound to a key or key-sequence. When you press Esc-Ctrl-D, MEmacs prompts for the key to describe. If you enter a key sequence, such as Ctrl-L or Esc-K, MEmacs will respond with the corresponding function. In this case, Redisplay and Reset-keys, respectively.

Keys are bound when they can be used to perform a function. For instance, any key, or key sequence, that can be used as a shortcut for a menu item is bound to that menu item.

Bind Key**Esc-Ctrl-B**

Allows you to bind a key to a function. When MEmacs prompts for the key to bind, enter the function (following the format used in the menu items) then the key or key sequence. To check if the key was bound properly, use the Describe key command (Esc-Ctrl-D).

Unbind Key**Esc-Ctrl-U**

Allows you to return a bound key to an unbound state. When MEmacs prompts for the key to unbind, enter the key or key sequence. MEmacs will then reply Key is not bound.

You cannot unbind the standard bound keys that are used as commands. If you use Unbind Key on a key that was not previously bound, you will not receive the Key is not bound message.

Echo**Esc-Ctrl-E**

Displays the string typed in the command line. This command is usually used when creating or editing executable MEmacs script files.

Move to Edge of Window**Shift-Cursor**

By holding down Shift and a cursor key, MEmacs will move the cursor to the top, bottom, left, or right edge of the screen. This is subject to the amount of text available.

Delete the Next Character**Ctrl-D**

Deletes the character at the current cursor position. This is the same as pressing Del.

Delete the Previous Character**Ctrl-H**

Deletes the character to the left of the current cursor position. This is the same as pressing Backspace.

Move to Next Line**Ctrl-M**

Inserts a newline character after the current cursor position and moves the cursor to the start of the new line.

Move x number of Characters**Ctrl-F****Ctrl-B**

Allows you to move the cursor forward or backward a specified number of spaces. The default value of this command is one character. However, you can establish a higher value by using Ctrl-U to set the argument value. Press Ctrl-F to move forward the specified number of characters, or press Ctrl-B to move backward.

Customizing MEmacs

When MEmacs is opened, it attempts to read the contents of an Emacs__pro file to see if there are any commands that it should automatically execute. This is a convenient way of saving commonly used commands, command sequences, or text strings. You can actually have several Emacs__pro files — a global file that is used every time MEmacs is opened and more specialized local files that are only used in certain instances. (The Emacs__pro file does not already exist; you have to create it.)

To create a global file of commands place the Emacs__pro file in the S: directory. Local files can be put in any directory. If that directory is the current directory when MEmacs is opened, the commands in that particular local file will be executed.

When both local and global Emacs_{pro} files are present, the local file overrides the global file.

For example:

```
Set Case On
Set-Key F11 "Dear Sirs:"
Set-Key F12 "^S Workbench"
Set-Key F13 "^X^B"
```

makes the following assignments:

- Shift-F1 Type the text string Dear Sirs:.
- Shift-F2 Search forward for the next occurrence of the
 word Workbench. (The Set Case On commands
 make any text searches case sensitive.)
- Shift-F3 Display the list of buffers.

Remember, you must use Ctrl-Q to enter a Ctrl-key sequence. For instance, to enter the ^S character shown in the example, you would have to press Ctrl-Q-Ctrl-S.

EDIT

EDIT processes multiple files line by line. EDIT moves through the input, or **source file**, passing each line, after any alterations, to a sequential output file, the **destination file**. An EDIT run, therefore, makes a copy of the source file that contains any changes you made with the editing commands.

Although EDIT usually processes the source file in a forward sequential manner, it has the capability to move backward a limited number of lines. This is possible because EDIT doesn't write the lines to the destination file immediately, but instead holds them in an **output queue**. The size of this queue depends on the amount of memory available. If you want to hold more information in the queue, you can use the OPT option of EDIT to increase the amount. (This is described in the following section, "Starting EDIT.")

You can make more than one pass through the text.

EDIT allows you to:

- change parts of the source
- output parts of the source to other destinations
- insert material from other sources
- edit files larger than the available memory

Starting EDIT

You must start EDIT through a Shell. The correct Format and Template are shown below:

Format: EDIT [FROM] <filename> [[TO] <filename>]
[WITH <filename>] [VER <filename>]
[[OPT P <lines> | W <chars>] |
[PREVIOUS <lines> | WIDTH <chars>]]

Template: FROM/A,TO,WITH/K,VER/K,OPT/K,
WIDTH/N,PREVIOUS/N

The FROM argument specifies the source file that you want to edit. You must specify a source file with EDIT although the FROM keyword is optional. Unlike ED, you cannot use EDIT to create a new file. If you attempt to create a new file, AmigaDOS displays an error stating that it cannot find the filename in the current directory.

The TO argument specifies the destination file to which EDIT will send its output, including editing changes. If you omit the TO argument, EDIT uses a temporary file. This temporary file is renamed as, and will overwrite, the FROM file when editing is complete.

The WITH keyword specifies a file containing editing commands.

The VER keyword specifies the file to which EDIT sends error messages and line verifications. If the VER argument is not given, EDIT uses the screen.

You can use the PREVIOUS and WIDTH options to increase or decrease the amount of available memory. The PREVIOUS option sets the number of previous lines available to EDIT to the integer <n>. The WIDTH option sets the maximum number of characters allowed on a line to <n>. EDIT multiplies the number of previous lines by the maximum number of characters (PREVIOUS * WIDTH) to determine the available memory. The default values are PREVIOUS 40 WIDTH 120.

You can also use `OPT P <n>` and `OPT W <n>` to specify the `PREVIOUS` and `WIDTH` options. However, do not use the `OPT` keyword with `PREVIOUS` and `WIDTH`.

Using Edit

This section explains how EDIT processes information. It explains:

- how input is handled
- what appears on the screen
- how output is handled
- the format of EDIT commands
- the types of arguments used by commands

The Current Line

As EDIT reads lines from the source file and writes them to the destination file, the line that EDIT is working on at any time is called the current line. When you start EDIT, the current line is the first line of the source file. Some things to keep in mind:

- Every command that you enter refers to the current line.
- All text changes are made to the current line.
- New lines are inserted before the current line.

EDIT identifies each line in the source file by a unique line number. This is not part of the information stored in the file. EDIT computes the line numbers by counting the lines as they are read. You can refer to a specific line by using its line number.

EDIT distinguishes between original and non-original lines. The original lines are the lines of the source file. A line that has been read retains its original line number as long as it is in

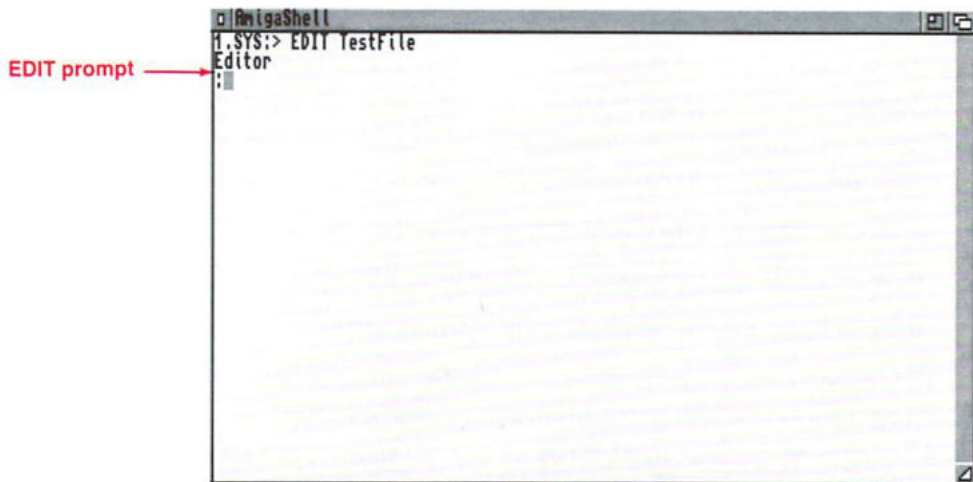
main memory, even if you delete or add lines before or after it. The line numbers remain unchanged until you renumber them with the REWIND or = commands, explained on page 9-83.

Any lines that are inserted into the source, or original lines that are split, are considered non-original lines and are not assigned line numbers.

You can only refer to original lines when using commands that take line numbers as arguments. EDIT moves forward, or backward, according to whether the line number you specify is before or after the current line. When searching for an original line, EDIT passes over non-original lines.

Prompts

When EDIT is ready to accept a command, it displays a prompt (:) and a cursor.



Usually, EDIT is run interactively, with the commands being typed at the keyboard and all error messages and line verifications appearing on the screen. Certain commands, such as those that change the information in a line, cause EDIT to

display the revised line after the command is executed. This is known as **line verification**. When a line is verified, the EDIT prompt does not appear. Instead, the cursor appears on the line below the verification. You can enter commands at the cursor; you don't have to have the prompt. However, if you press Return, the prompt will reappear.

The following circumstances cause line verification:

- When you type a new line of commands for a new current line that EDIT has not yet verified or for a current line that has changed since the last verification.
- When EDIT has moved past a line that it has changed but not yet verified.
- When EDIT displays an error message.

Output Processing

EDIT sends revised lines and any information inserted into the file to the output, or destination, file specified by the TO argument. If no file is specified, EDIT uses a temporary file that is later renamed as the source file. EDIT does not send the edited lines to the destination file immediately. Instead, the lines are kept in an output queue in main memory. The number of lines that can be held in this queue is determined by EDIT's PREVIOUS option. (The default is 40 lines.)

Lines are written to the destination file as the output queue hits its maximum number of lines. For instance, if the output queue contains 40 lines and a new line is sent to the queue, the oldest line in the queue is written to the destination file. Until EDIT has actually written a line to the destination file, you can move back and make it the current line again.

When EDIT reaches the last line of a source file, it creates an extra blank line at the end of the file. This line has a line number one greater than the number of lines in the file. EDIT verifies the line by displaying the line number and an asterisk.

If this extra line is the current line, any commands to change the line or to move forward in the file will result in an error.

Commands

An EDIT command is either a sequence of letters or a single special character, such as #. If the command consists of more than four letters, only the first four letters of the name are significant. For instance, You only need to type REWI to execute the REWIND command. Commands may also be followed by arguments.

Spaces between a command and the first argument, between non-string arguments, and between commands are optional. A space is only mandatory when it must separate two successive items that could otherwise be treated as one, such as two numbers or an alphabetic argument following an alphabetic command.

EDIT commands are not case-sensitive. They can be typed either uppercase or lowercase.

You can enter a command in three ways:

- typing the command(s), then pressing Return.
- typing the final command argument, then pressing Return.
- typing a semicolon or closing parenthesis.

Arguments

There are four different types of arguments you can use with EDIT commands:

- strings
- qualified strings
- numbers
- switch values

Each of these is explained in the following sections, and you'll learn more about the arguments as you encounter the command sections.

Strings

A string is a sequence of up to 80 characters enclosed in **delimiters**. A delimiter is a character that indicates the beginning or end of a string. You can use any common English punctuation characters, except the semicolon, and the four arithmetic operators as delimiters. Acceptable examples are:

`/ + - , ? : *`

The delimiter cannot appear in the string. For instance, you can't use apostrophes to delimit a string that contains an apostrophe. For example:

`'it's'`

will be recognized as two strings: `it` and `s` that are delimited with apostrophes. However:

`/it's/`

will be recognized as one string: `it's`.

The final delimiter can be omitted if it is at the end of the command line.

Below are some examples of strings:

Delimited String	String
<code>/A/</code>	<code>A</code>
<code>?Hello?</code>	<code>Hello</code>
<code>+String without final delimiter</code>	<code>String without final delimiter</code>

Commands that take two string arguments use the same delimiter for both strings and do not repeat it between the arguments. The second string always specifies replacement

text. An example of this is the A command. The A command inserts its second string after the first occurrence of the first string. For example:

```
A /Jingle/all the way
```

inserts all the way after the first occurrence of Jingle. (Remember that the final delimiter does not have to be typed.)

Strings are case-sensitive. In the above example, EDIT would look for Jingle with a capital J. If EDIT only found jingle an error message would be displayed.

You can also use an empty, or **null**, string. Null strings are commonly used with exchange commands to delete a string of text. The E command replaces the first string with the second string. For example:

```
E /slay/sleigh
```

replaces slay with sleigh. However, if you typed:

```
E /slay//
```

slay would be replaced with the null string. In other words, slay would be deleted from the current line. If you were to specify a null string after the A command, nothing would happen because you've asked EDIT to insert nothing after the first string.

Qualified Strings

In some cases, you may want to search for a string by its context, such as whether it appears in the beginning or end of a line. In this case, you must use a **qualified string** with the command. A qualified string is an argument that is preceded by a **qualifier**, one or more letters specifying the context.

The available qualifiers are:

- B The string must appear at the beginning of the line; cannot be used with E, L, or P. If B appears with the null string, it matches with the beginning of the line since it is not told to look for anything.
- E The string must be at the end of the line; cannot be used with B, L, or P. If E appears with the null string, it matches with the end of the line.
- L EDIT will search for the string from the end of the line to the beginning of the line (instead of from right to left). If there is more than one occurrence of the string in a line, L makes sure that the last one is found. L cannot be used with B, E, or P. If L appears with the null string, it matches with the end of the line.
- P The line must match the string precisely and must not contain any other characters. P cannot be used with B, E, or L. If P appears with a null string, it matches an empty line.
- U The search is not case-sensitive. It will match any occurrence of the string regardless of whether it occurs in uppercase, lowercase, or a combination of the two.

Numbers

Line numbers are a special form of number and must always be greater than zero. A period represents the current line and may be used instead of the line number. An asterisk represents the last line at the end of the source file. For example, the M command takes a line number as its argument and makes that line number the current line. So:

M *

instructs EDIT to move to the end of the source file.

Switch Values

Some EDIT commands must be “switched” on or off. In this case, the command takes a single character, either + or -, as an argument. The plus sign turns the command on, and the minus sign turns a command off.

Multiple Commands

You can repeat a command by typing a number in front of it. For example, the N command allows you to move forward to the next line in the source file. Typing 4N moves you ahead four lines, essentially repeating the N command four times.

If you use a number before a command that cannot be repeated, it will simply execute once.

You can enter more than one command on the same line. You do not need to put spaces between the commands unless the two successive commands could be mistaken for one item. You must separate the commands with a semicolon if a command has a variable number of arguments, such as an exchange command, and the next command could be mistaken as the previous command’s argument.

To repeat a series of commands, enclose the entire line in parentheses. For instance:

```
6(E/;/ N)
```

Will exchange the next six occurrences of a colon with a semicolon.

Command groups may not span more than one line of output. If you type a command group that is longer than one line, EDIT will only accept the commands up to the end of the first line. Then, because EDIT does not find a closing parenthesis at the end of that line, it displays the following error message:

```
Unmatched parenthesis
```

If you put a 0 before a command or a command group, it will be repeated indefinitely or until EDIT reaches the end of the source file.

EDIT Commands

This section explains the EDIT commands. The text conventions used are listed below:

- Command names are shown in uppercase although they do not have to be entered that way.
- Angle brackets indicate that information must be substituted. For example, `<string>` indicates that the command takes a string argument.
- Square brackets indicate that the argument is optional. For example, `[<n>]` indicates that the command can take an optional numeric argument.
- An `<n>` represents a numeric argument.
- Slashes are used as delimiters for strings.
- Periods are used as delimiters for filenames (slashes cannot be used since they are used to separate filenames).

Selecting the Current Line

The commands in this section let you move through the file and select the current line.

Move to a specific line number

M `<n>`

The M command allows you to select a new current line by specifying its line number. M takes a line number, period, or asterisk as its argument. Only original lines can be accessed by line number.

- M <n> Moves to line <n> provided it is still in main memory and makes it the current line.
- M + EDIT moves through all the lines currently held in memory until it reaches the last one. This last line is then made the current line.
- M - Makes the last line in the output queue the current line. This is like telling EDIT to move back as far as possible in main memory.

Move to the next line in the source file **N**

If you give a number before the N command, you can move that number of lines forward.

- N Moves to the next line and displays the line.
- 4N Moves four lines forward.

If you give the N command when on the last line of the source file, EDIT creates an extra line at the end of the file. However, if you try to use an N command when you are already on this extra line, EDIT displays an error message, such as Input Exhausted.

Move to the previous line in the source file **P**

You can move more than one line back by repeating P or by giving a number before the P command.

- P Moves to the previous line.
- 4P Moves back four lines.

It is only possible to go back to previous lines that EDIT has not yet written to the destination file. EDIT usually lets you go back 40 lines, unless this has been changed with the PREVIOUS option.

You can combine the M command with N or P. For example:

M12; 3N

Moves you to line 12 of the file, then up 3 more lines.

Find

F <string>

The F command allows you to select a current line by specifying some of its content. For example:

```
/bells/
```

finds the next line containing `bells` and makes that the current line. The search begins at the current line and moves forward through the source file until the required line is found. If EDIT reaches the end of the source file without finding a matching line, it displays the message `Source Exhausted`.

You can use qualifiers with the F command to search for text in a specific context. For example:

```
FL /bells/
```

restricts the search to the end of the lines.

Search Backward

BF <string>

The BF command allows you to look backward through the source file for a line containing the specified string. For example:

```
BF /sleigh/
```

searches backward for the closest line containing `sleigh`.

BF starts at the current line and moves backward towards the beginning of the file. If EDIT reaches the start of the output queue without finding a matching line, it displays the message `No More Previous Lines`.

Editing the Current Line

The commands in this section let you add new material, or replace material, on the current line.

Insert <string2> after <string1> A <string1> <string2>

The A command inserts the second string after the first occurrence of the first string.

For example, if the current line contained:

What fun it is to sing

then the following command:

A /to/laugh and/

would change it to:

What fun it is to laugh and sing

If the first string is a null string, EDIT inserts the second string at the beginning of the line. You can use qualifiers to further restrict the context of the first string. If the first string cannot be found on the current line, a No Match error is given.

Insert <string2> before <string 1> B <string1> <string2>

The B command inserts the second string before the first occurrence of the first string. For example, if the current line reads:

In a open sleigh

the following command:

B /open sleigh/one horse/

would change it to:

In a one horse open sleigh

If the first string is a null string, EDIT inserts the second string at the beginning of the line. You can use qualifiers to further restrict the context of the first string. If the first string cannot be found on the current line, a No Match error is given.

Exchange <string2> for <string1> E <string1> <string2>

The E command replaces the first occurrence of the first string with the second string. For example, if line 3 reads:

Oh what fun it is to slide

The command:

```
E /slide/ride/
```

would change it to:

```
Oh what fun it is to ride
```

To delete parts of a line, make the second string a null string.
For example:

```
E /to ride//
```

would change the line to:

```
Oh what fun it is
```

If the first string is a null string, the second string is inserted at the beginning of the line. If the first string cannot be found on the current line, a No Match error is given.

Inserting and Deleting Lines

The commands in this section let you insert new material (non-original lines) and delete lines from the source file. You can also insert complete files into the file.

Insert one or more lines

I [<n>]

You can give the I command alone or with a line number, period or asterisk. If given alone or followed by a period, EDIT inserts the text before the current line. If given with an asterisk, the text is inserted at the end of the file.

To indicate the end of your insertion, press Return, type Z, and press Return again. For example:

```
I 8
Laughing all the way
Z
```

inserts the line before line 8. Line 8 then becomes the current line, as the newly inserted line is considered non-original and is not assigned a line number.

If a filename is given after the I command, the contents of the specified file are inserted before the current line.

If a line number is not specified after the I command, the new information is inserted above the current line.

Delete one or more lines **D [<n>]**

To delete the current line, type D with no arguments. The following line will become the new current line. To delete a single line, specify the line number after the D command:

D 8

deletes line 8. Line 9 will be the new current line.

To delete a range of lines, specify the lines numbers (inclusive) after D:

D 9 20

deletes line 9 through to and including line 20. Line 21 will be the new current line.

To delete everything from the current line through to the end of the source file, type:

D . *

Delete all lines until the specified string is found **DF <string>**

The DF command tells EDIT to delete successive lines from the source file until it finds a line matching the given string. That line then becomes the new current line. A DF command with no argument, searches for the last string entered, deleting all lines until it finds it.

Delete existing lines and replace with new text **R [<n>]**

The R (Replace) command lets you delete lines then insert new ones. This is equivalent to using the D command, followed by the I command.

To replace the current line, type:

```
R
<replacement text>
Z
```

To replace one line, type the line number after R. For instance:

```
R5
<replacement text>
Z
```

deletes the existing text in line 5, replaces it with the specified text, and makes line 6 the current line.

Change the terminator

Z <string>

The **terminator** is a command that tells EDIT that it has reached the end of any new text that is being inserted. As shown above, the default command is Z. However, you can change this by specifying a string after the Z command. The string can be up to 16 characters and it is matched regardless of the case of its characters. For example:

```
Z /the end/
```

changes the terminator to the end. If you were entering an I command, you would have to type:

```
I
<new text>
the end
```

Show current information about EDIT

SHD

The SHD (Show Data) command displays saved information values, such as the last string searched for, the last command entered, and the input terminator (usually Z unless changed by the user).

Turn trailing spaces on/off

TR +|-

The TR command allows you to suppress any blanks that fall at the end of lines. By default EDIT suppresses blanks (the TR command is turned off). To preserve any end of the line blanks in both the input and output lines, type:

```
TR +
```

Editing Line Windows

Usually EDIT acts on the entire current line. However, you can define subsections of the line on which EDIT will execute all subsequent commands. These line segments are called **line windows**. In the descriptions of EDIT qualifiers, the beginning of the line always means the beginning of the line window.

Whenever EDIT verifies a current line, it indicates the position of the line window by displaying a > character directly beneath the line. For example:

```
1.
  Jingle bells, jingle bells
    >
```

the line window contains the characters to the right of the **character pointer** — ells, jingle bells. EDIT omits the pointer if the line window begins at the start of the line.

The following commands control the position of the character pointer:

>	Moves the pointer one character to the right.
<	Moves the pointer one character to the left.
PR	Resets the pointer to the start of the line.
PA <string>	Moves the pointer to the first character after the specified string.
<string>	Moves the pointer to the first character before the specified string.

The following commands change the character at the current pointer, then move the pointer:

\$	Makes the character at the pointer lowercase, then moves the pointer one character to the right.
%	Makes the character at the pointer uppercase, then moves the pointer one character to the right.

- The — (underscore) command deletes the character at the pointer, making it into a space, then moves the pointer one character to the right.
- # Deletes the character at the pointer, then moves the rest of the line one character to the left. To delete several characters, specify a number before the #. For example:
5#
deletes the next five characters in the window.

You can use a combination of the above commands to edit a line character by character.

Some other commands let you insert and exchange text on the current line, similar to the A, B, and E commands explained earlier. However, when the operation is complete, the character pointer is moved.

Insert <string2> after <string1> AP <string1> <string2>

The pointer is moved after <string2>.

Insert <string2> before <string1> BP <string1> <string2>

The pointer is moved after <string2>.

Exchange <string1> with <string2> EP <string1> <string2>

The pointer is moved after <string2>.

Delete Till After DTA <string>

Deletes all the text from the beginning of the line or the character pointer to end of the specified string.

Delete Till Before DBA <string>

Deletes all the text from the beginning of the line or the character pointer stopping just before the specified string.

Delete From After **DFA** <string>

Deletes all the text from just after a specified string to the end of the line.

Delete From Before **DFB** <string>

Deletes all the text starting with the specified string to the end of the line.

Splitting and Joining Lines

The commands in this section let you split a line into more than one line and join together two or more successive lines.

Split line before <string> **SB** <string>

The SB command splits the current line before the specified string. EDIT sends the first part of the line to the output queue. The remainder of the line is made into a new, non-original current line. For example, if the current line is:

Bells on bob-tail ring, making spirits bright

The command:

SB /making/

splits the line before making. Making spirits bright will become the new current line.

You can also use qualifiers with SB to restrict the context of the string.

Split line after <string> **SA** <string>

The SA command splits the current line after the specified string. EDIT sends the first part of the line to the output queue. The remainder of the line becomes the new current line.

You can use qualifiers with SB to restrict the context of the string.

Join two lines**CL [<string>]**

The CL (Concatenate Lines) command joins the current line with the next line of the source file. The <string> argument is optional. However, if a string is specified, it will be added to the end of the current line, then that entire line will be joined with the next line in the source file. For example, if the current line and following line are:

A sleighing
tonight!

the command:

CL /song/

will add song to the end of the current line and join it with the subsequent line. The result will be:

A sleighing song tonight!

Renumbering Lines

The commands in this section let you renumber the lines of the source file to include non-original lines and to update a file that has been heavily edited.

Renumber source lines**= <n>**

The = command sets the current line number to <n>. If you then move to the lines below <n>, EDIT rennumbers all the following original and non-original lines. However, if you move from <n> to previous lines, EDIT marks all the previous lines in the output queue as non-original.

Return to the beginning source file**REWIND**

The REWIND command moves back through the source file so that line 1 becomes the current line. EDIT scans the rest of the source file, then writes the lines to the destination file. The destination file is then closed and re-opened as a new source file. Any non-original lines will now be recognized as original lines.

You do not have to type the complete word "REWIND"; the first four letters suffice.

Verifying Lines

Normally, EDIT is operating in an interactive state and is verifying lines as a result of various commands. The commands in this section describe different ways of verifying lines.

Turn Verification on/off

V + | -

The V command allows you to turn off line verification, the lines will not be displayed on the screen. To turn verification off, type:

V -

To turn it back on, type:

V +

Verify the current line

?

The ? command allows you to verify the current line. The line number and the contents of the line will be shown on the screen.

Verify the current line with character indicators

!

If a binary file is being edited, non-graphic characters will be represented with question marks (?). The ! command produces two lines of verification. In the first line, EDIT replaces all non-graphic characters with the first character of their hexadecimal value. In the second line, EDIT displays a minus sign under all the positions corresponding to uppercase letters and the second hexadecimal digit in the positions corresponding to non-graphic characters. All other positions contain space characters.

Inspecting the Source File

The following commands tell EDIT to advance through the source file, sending the lines it passes to the verification file as well as to the normal output. These commands are known as type commands because they allow you to display lines on the screen. The lines are also passed to the output queue. After the last line is typed, it becomes the new current line.

Type <n> lines to the screen **T<n>**

The T command types the specified number of lines to the screen. The first line typed is the current line.

If you omit the <n>, typing continues until the end of the source file. You can interrupt the command by pressing Ctrl-C.

Type the lines in the output queue **TP**

The TP (Type Previous) command displays the lines currently held in the output queue.

Type until EDIT has replaced all the lines in the output queue **TN**

The TN (Type Next) command types from the current line forward until all the lines in the output queue are replaced. In other words, if the output queue holds 40 lines and the current line is 60, TN will type from line 60 through to line 100. Lines 60-100 will now be in the output queue. The previous contents (lines 20-60) are sent to the destination file.

Type with line numbers **TL <n>**

The TL command is similar to the T command in that it types the specified number of lines. However, TL also displays the line numbers. Inserted and split lines do not have line numbers, so EDIT displays + + + + instead.

Remember that you can always use the + command to renumber non-original lines.

Making Global Changes

Global changes are changes that take place automatically as EDIT scans the source file in a forward direction. You can start and stop global changes with the commands described in this section.

The following commands automatically apply an A, B, or E command, as appropriate, to any occurrence of <string1> in a new current line. They also apply to the current line that is in effect when the command is given.

GA [qualifier] <string1> <string2>

GB [qualifier] <string1> <string2>

GE [qualifier] <string1> <string2>

For instance, if you want to change DF0: to DF2: throughout an entire file, type:

```
GE /DF0:/DF2:/
```

Cancel a global command

CG [<id number>]

The CG command cancels a global command. When a global operation is set up with the GA, GB, or GE command, the operation is given an identification number. This identification number, such as G1, is output to the verification file (or the screen if EDIT is interactive).

If no argument is given with CG, all global operations are cancelled. To only cancel a specific operation, specify the identification number after the CG command.

Suspend a global command

SG [<id number>]

The SG command suspends a global command. If no argument is given, all global operations are suspended. To only suspend a specific operation, specify the identification number.

Enable a global command**EG [<id number>]**

The EG command resumes a global operation that had been suspended with the SG command. As with the other global commands, unless a specific identification number is specified, all global commands will be resumed.

Show global commands**SHG**

The SHG command displays the current global commands and their identification numbers. It also gives the number of times each global search string was matched. For example:

```
:shg
1 3 GE /DF0:/DF1:/
```

Changing Command, Input, and Output Files

The following section describes commands that can change the files that you set up when you started EDIT from the Shell. These files are:

- the command file — started with the WITH option
- the input file — the source file specified with FROM
- the output file — the destination file specified with TO

Changing the Command File**C <filename>**

The C command lets you read EDIT commands from a specified file. Since AmigaDOS uses a slash (/) to separate filenames, use a different character, such as a period or question mark, to delimit the file. For example:

```
C .:T/xyz.
```

reads the commands from the XYZ file stored in the T directory.

When EDIT has executed all the commands in the specified file, it closes the file. You can then enter commands through the keyboard.

Changing the Input File**FROM <filename>**

The FROM command lets you read lines from another source file. For example:

```
FROM ./S/Script.
```

allows you to read lines from the Script file in the S directory. The current line remains current and is read from the original source file; however, the next line will be read from the Script file.

EDIT does not close the original source file. You can reselect the source file by entering the FROM command without an argument.

Closing a File**CF <filename>**

The CF command lets you close the destination file that you originally specified with the TO command. You can then open that file for input. You can also use the CF command to close a new input file that you had opened.

If you close a file, then reopen it, EDIT starts reading from the first line of that file, not from the line that it was on when you closed it.

You should always close files that you are finished working with so that the memory used by those files can be used by the system.

An example of using the FROM and CF commands is shown below:

Command	Action
M10	Pass lines 1-9 in the original source file to the output queue.
FROM .XYZ.	Select the XYZ file for new input; line 10 of the original source file remains current.

M6	Pass line 10 from the original file, then pass lines 1- 5 from the XYZ file to the output queue. Line 6 of XYZ is the new current line.
FROM	Reselect the original source file.
M14	Pass line 6 from XYZ, then lines 11-13 from the original source file to the output queue. Line 14 of the source file is the new current line.
FROM .XYZ.	Reselect file XYZ. Line 14 of the source file is still the current line.
M*	Pass line 14 of the source file and all remaining lines of file XYZ to the output queue. An extra line will be added to the end of file XYZ. That line will be the new current line.
FROM	Reselect the original source file. The extra line added to file XYZ will still be the current line.
CF.XYZ.	Close file XYZ.
M*	Pass the remaining lines of the source file (lines 15 to the end of the file) to the output queue.

Changing the Output File

TO <filename>

The TO command lets you specify a different file as the destination file. When EDIT executes a TO command, it writes out the existing queue of output lines to the new TO file.

EDIT will continue to use the new TO file until another file is specified. To reselect the original destination file, give the TO command with no argument. Although the alternate output file will not be used, it will remain open. To add additional lines to it, select it again with the TO <filename> command.

Command	Action
---------	--------

M11	Passes lines 1-10 of the source file to the original destination file.
TO .XYZ.	Makes XYZ the new output file.
M21	Passes lines 11-20 to file XYZ.
TO M31	Makes the original destination file current, and passes lines 21 to 30 to it.
TO .XYZ.	Makes XYZ the current output file.
M41	Passes lines 31 to 40 to XYZ.
TO	Makes the original destination file current.

These input/output commands are useful when you want to move part of the source file to a later place in the output. For example:

Command	Action
---------	--------

TO .XYZ.	Sends the output queue to file XYZ.
1000N	Advances through the next 1000 lines of the source file.
TO	Selects the original destination file.
CF.XYZ.	Closes the XYZ file.
I2000 .XYZ.	Inserts the 1000 lines from the source file that were sent to file XYZ back into the source file above line 2000.

Stop executing the command file**Q**

The Q command stops EDIT from executing the current command file specified with the WITH keyword or with the C command. EDIT reverts to any previous command file. A Q at the outermost level is equivalent to the W command.

Ending EDIT

The commands in this section explain how to exit EDIT.

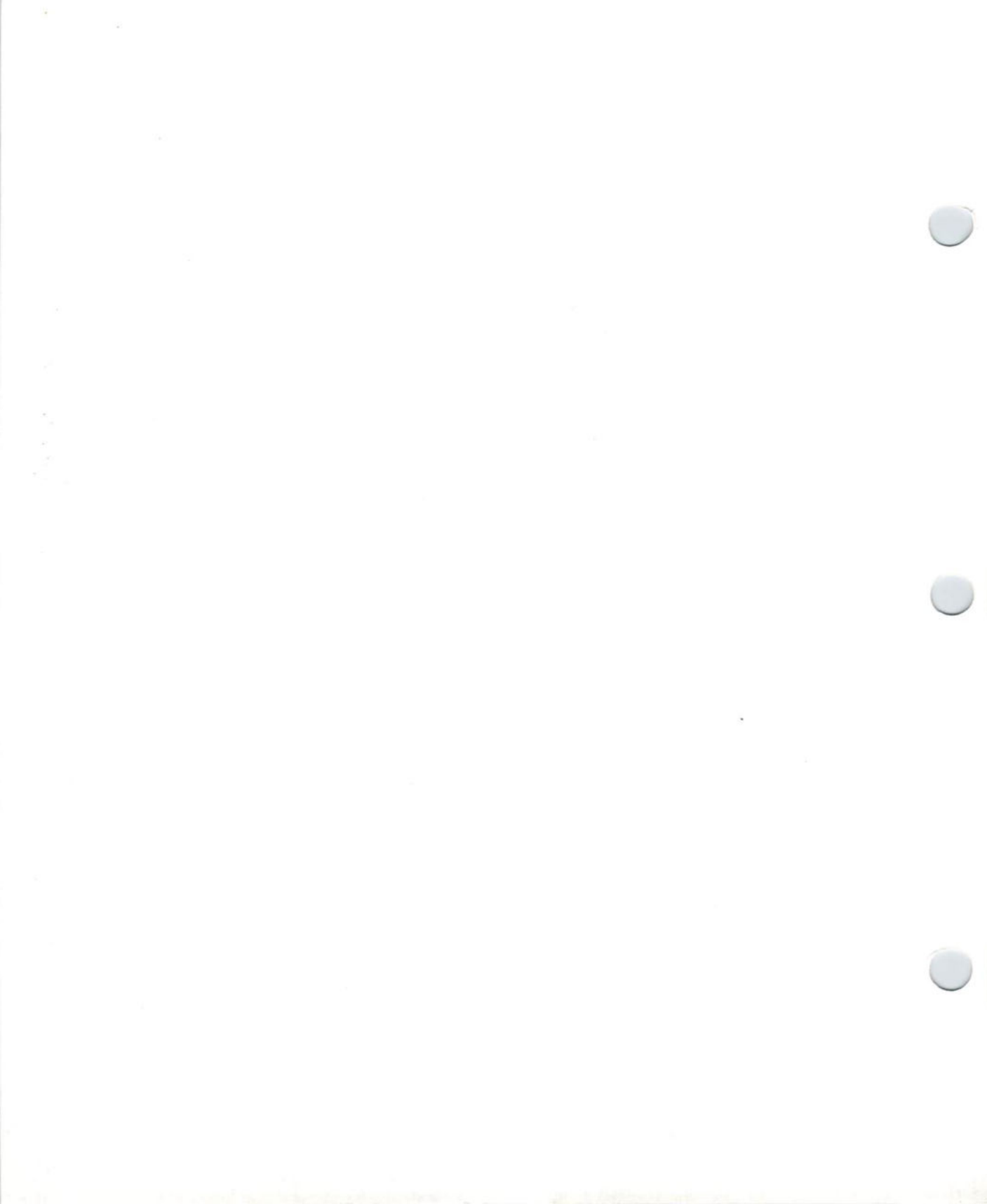
Exit, saving changes**WINDUP**

The W (Windup) command exits EDIT, saving all changes to the destination file specified by TO. EDIT exits when it has reached the end of the source, closed all the files, and relinquished the memory.

If you started EDIT without specifying a destination file, EDIT renames the temporary destination file it created with the same name as the original source file. It renames the original source file as :T/Edit-backup. This backup file is only available until the next time you run EDIT.

Exit, without saving changes**STOP**

The STOP command stops EDIT immediately without saving any changes to the source file. STOP prevents EDIT from overwriting the original source file, ensuring that no changes are made to the original input information.



Chapter 10. AREXX Programming Language

Introduction

AREXX is a programming language designed to offer flexibility to customize your working environment. AREXX acts as a central hub through which applications may send data and commands to each other. This allows software created by different companies to interact and, in turn, allows the user to create custom applications by integrating off-the-shelf software products. For example, with AREXX it is possible to set up a telecommunications package to dial an electronic bulletin board, download financial data, and then pass that data to a separate spreadsheet for statistical analysis automatically, without user intervention.

AREXX is also useful in writing small programs, called scripts or macros, that allow the automation of repetitive tasks. A script or macro can be written to automate a telecommunications program login or to transfer text from a wordprocessor into a desktop publishing program.

The complete AREXX programming language is provided on the floppy disks included with this manual. The AREXX software is also written to your Amiga's hard disk (if applicable).

Who is AREXX For?

AREXX is for the user who has become well acquainted with the Amiga. A basic understanding of both Workbench and AmigaDOS will aid in the understanding of how AREXX works and how it can be put to use. Chapters 1 through 9 cover the fundamentals of Workbench and AmigaDOS.

AREXX is used to customize the Amiga's working environment, and one must be familiar with that environment before it can be tailored to suit one's needs. Using ready-made AREXX programs and scripts does not require previous Amiga experience, but a working knowledge of the Amiga's operation will permit you to change the scripts to suit your needs.

Experienced Amiga users will find that AREXX can provide more control over their working environment. In many cases AREXX is easier to work with and more powerful than AmigaDOS, and AREXX can be used to enhance or replace pre-existing AmigaDOS commands and scripts. AREXX can also be used to tie applications together to create a more efficient employment of resources in the form of integrated applications.

AREXX gives the Amiga a programming language that can be universally understood by both AmigaDOS and applications run on the Amiga. This is an important advancement for both the development of Amiga software and the Amiga programmer. Software developers, who understand the power and flexibility of AREXX, will recognize the advantage in adding an AREXX port to their applications.

What Do You Have to Know to Use AREXX?

Before using AREXX, you should know a few things about the Amiga and its operation. You need to know how to:

- 1) Open a Shell and enter AmigaDOS commands (See Chapter 7 for full details.)
- 2) Use a text editor (There are several editors included with your Amiga software. Information on editors can be found in Chapter 9.)
- 3) Create a User-startup file (The User-startup is covered in Chapter 7 and is created using one of the editors covered in Chapter 9.)

Chapter Organization

This chapter will introduce you to AREXX, tell you how to begin using it, and provide a reference section of AREXX commands. This chapter is arranged in the following order:

- Introduction — Gives an overview of AREXX, its implementation on the Amiga, and tells you how to get started using it.
- Elements of AREXX — Introduces the language structure and syntax. (Page 10-26)
- Instructions — Describes the action statements of AREXX. (Page 10-50)
- Commands — Describes the program statements of AREXX. (Page 10-74)
- Functions — Explains how functions are called and documents the Built-in function libraries. (Page 10-82)

- Support library Functions — Details how to open the AREXX library and describes each supported function. (Page 10-129)
- Tracing and Interrupts — Describes the source-level debugging features used in the development and testing of programs. (Page 10-134)
- Parsing and Templates — Describes the instructions used to extract patterns of information from strings. (Page 10-146)
- REXXC Directory — Includes descriptions of the command utility programs contained in this directory. (Page 10-154)
- Error Messages — Lists the error messages issued by the interpreter. (Page 10-157)

Further information on learning and using AREXX can be found in the following publications:

Modern Programming Using REXX, by R. P. O'Hara and D. G. Gomberg, Prentice-Hall, 1985

The REXX Language: A Practical Approach to Programming, by M. F. Cowlishaw, Prentice-Hall, 1985.

In this chapter, AREXX instructions, functions and commands appear in **boldface** type. This is to distinguish them from the rest of the text. These words are not included in the Glossary at the end of this manual.

AREXX on the Amiga

Because AREXX can tie applications together and have them communicate with each other, it has been made an integral part of the Amiga operating system beginning with the release of V2.0. Specifically, AREXX uses two important features in the Amiga's operating system to allow creation of customized integrated software: multitasking and interprocess communication.

Multitasking

Multitasking is the ability to run more than one program at a time. This ability is built into the Amiga. For example, you can simultaneously edit a file, format a disk, and adjust your screen's colors.

Interprocess Communication

Interprocess communication (or IPC for short) refers to a computer's ability to allow the exchange of information between currently operating programs. This is efficient as it allows the updating of information without tedious steps. For instance, the Amiga makes it possible to transfer information between two programs which are concurrently running.

Interprocess Communication and Ports

Interprocess communication occurs through message ports attached to each program. A message port is an address contained in an application that can receive and send messages. A message sent to AREXX from an application will be directed by the Amiga operating system to the AREXX message port. A message sent from AREXX to an application occurs in a like manner. Each message port has a name (usually the name of the program), and sending a message requires the use of the port's name in an AREXX script.

The order of operations in the sending and receiving of a message is briefly described below:

- 1) An application opens a message port
- 2) The application waits to receive a message
- 3) The Amiga lets the application know that a message has arrived
- 4) The application acts on the message
- 5) The application lets the message's sender know that the message has been received and processed

These five steps are critical in interprocess communication, especially the final step of letting the sender know that the message has been received and processed.

Multitasking and Interprocess Communication Together

Multitasking and interprocess communication are critical in the creation of customized integrated software. Without the Amiga's ability to run several programs at once, and have these programs talk to each other, the idea of customized integrated software would remain just an idea. The final link is AREXX. AREXX allows you to control and modify how specific programs run in order of operation and how these programs communicate with each other. By writing simple AREXX scripts, you can gain control and flexibility in the use of applications. This flexibility will allow you to specifically tailor your working environment.

What is AREXX?

AREXX is the Amiga version of the IBM Rexx programming language. AREXX, like Rexx, is a powerful yet easy-to-use programming language that is especially useful as a scripting language. A script is a small program that instructs the operating system to perform a series of actions. Scripting languages are used to control and modify applications and to direct how they interact with other applications.

AREXX is an interpreted language. Interpreted languages are written in simple, ASCII characters and are composed one line at a time. This makes them easy to learn and work with. The interpreter in AREXX is RexxMast. If RexxMast finds an error while translating or executing a line, it will halt execution and return an error message. This interactive testing is both a learning tool and an aid in the production of programs. It will tell you exactly where an error has occurred.

Starting AREXX on the Amiga

RexxMast (the AREXX interpreter) is initialized when you boot your Amiga (this includes both hard and floppy disk boots). From this point AREXX can be run in two ways: automatically or manually.

Automatically

If you have a factory-installed hard disk drive, the AREXX software is written to your hard drive, and RexxMast is started when you boot your Amiga. To begin using AREXX, simply go into a text editor and start entering your program.

Manually

If the necessary AREXX files have been deleted from your Startup-sequence, you will be starting AREXX manually. There are two ways to start AREXX manually: through the Shell or through the RexxMast icon.

Starting AREXX Through the Shell

To start AREXX manually through the Shell, open a Shell and enter:

```
RexxMast
```

A message will appear notifying you that RexxMast is now running. To start using AREXX, simply go into a text editor and begin entering your program or script.

Starting AREXX by Icon

To start AREXX manually by icon, open the System drawer and double-click on the RexxMast icon. A message will appear notifying you that RexxMast is now running. To start using AREXX, simply go into a text editor and begin entering your program or script.

Setting AREXX to Start Automatically

AREXX can be set to start automatically by modifying your User-startup file (information on modifying files is located in Chapter 7) and assigning REXX (details on assigns can be found in Chapter 8) using the ED editor (details on the ED editor are located in Chapter 9). Here's how you do it:

- 1) Open a Shell and enter:

```
ed s:user-startup
```

An empty window called user-startup will appear.

- 2) Enter the following lines into the User-startup:

```
assign REXX: S:  
rexcmast >NIL:
```

- 3) When the Amiga is first turned on it looks for a path (details on paths can be found in Chapter 7) of directories. This path tells the Amiga where the boot files are. Enter the following path exactly as shown below:

```
Path sys:system sys:rexxc add
```

- 4) Save this User-startup file and exit ED. AREXX will now start automatically and be ready for immediate use each time you boot your Amiga.

Example #1

The following AREXX Terms are used in Example #1:

```
/* comment line */  
if  
then  
~ (tilde)  
open  
CLOSE  
exit  
call  
writeln  
readln
```


Example #1 is entered using a text editor and then executed through the Shell by entering the program name, preceded by the rx command. Now use MEmacs to enter the example exactly as shown; a discussion of the example follows. Incorrectly entering this example will cause at least one error message to appear.

Open a Shell and, at the command prompt, enter:

```
memacs rexx>window.rexx
```

By entering the information exactly as it is shown, the AREXX program will be named Window.rexx, and the MEmacs editor will appear ready to accept this AREXX example. Note that the first line of this program is a comment line which tells AREXX that the following information is part of an AREXX program named Window.rexx. **Do not enter the numbers to the left of each line. They are intended only as reference for the discussion that follows.**

1. /* Window.rexx This is Example #1 */
2. /* prepare window, but exit if we can't open it */
3. "if ~ open"('console', 'con:0/0/640/200/RexxWindow/CLOSE', 'W')
4. then exit 20
5. /* Write some text into the window */
6. call writeln 'console', 'Hello, world'
7. call writeln 'console', 'Press RETURN to exit'
8. /* Wait for user to enter anything; discard result */
9. call readln 'console'
10. /* All done. Ordinarily we would "close 'console'" and then
11. "exit", but AREXX does that automatically.
12. */

Once the above example is entered exactly as listed, recheck it just to make sure, especially the instances of spacing and single or double quotes.

EXAMPLE #1 Window.rexx — What each program line means

1. /* Window.rexx */

This is the AREXX comment line, with which all AREXX programs begin. AREXX will not execute any program that does not begin with a comment line. Note that the comment line begins with a /* and ends with a */.

2. /* prepare window, but exit if we can't open it */

Line 2 is another comment line that tells what will happen in Lines 3 and 4.

3. if ~ open('console','con:0/0/640/200/RexxWindow/CLOSE','W')

Line 3 illustrates a common programming function: the if . . . then conditional statement. The program asks AREXX to open a window with the given specifications and to treat this window as a file which will be written to. The ~ (tilde) character in front of open is an operator that literally translates to a "logical NOT." This means the statement should read, "if you cannot open . . ."

4. then exit 20

Line 4 is a continuation of Line 3. It serves as the second half of the if . . . then conditional statement, namely if AREXX cannot open a window then it will stop execution and return an error code of 20.

5. /* Write some text into the window */

Again this is a comment line. Note that AREXX doesn't pay any attention to comments, but it does require that the program begin with a comment line.

6. call writeln 'console', 'Hello, world'

Call is an instruction which begins executing a function and disregards anything the function may return. The function `writeln` tells AREXX to display the phrase 'Hello, world' in the window created by `open` in Line 3.

7. `call writeln 'console', 'Press RETURN to exit'`

Line 7 works similarly to Line 6, but displays 'Press RETURN to exit.'

8. `/* Wait for user to enter anything; discard result */`

Line 8 is another comment line.

9. `call readln 'console'`

Call is an instruction that requests the function `readln`. AREXX then waits for the user to press RETURN, and reads RETURN from the console.

10. through 12.

Lines 10. and 11. are comment lines. Line 12. ends with the `*/` characters; these three lines are not required for proper execution.

Since the program was named prior to it being entered in MEmacs, all that needs to be done is to save and exit. Simultaneously press Ctrl X and then Ctrl F to save and to exit (see Chapter 9 for more MEmacs commands.) This will save the program named `Window.rexx`, exit MEmacs, and return to the Shell command prompt.

Displaying Output

AREXX Terms

rx

.rexx

To display the output of this program, enter the following at the Shell command prompt:

```
rx Window.rexx
```

The output will appear in a window named REXXWindow. The output is:

```
Hello, world
```

```
Press RETURN to exit
```

When you press RETURN, the window will close and the program will exit. This program can be run in the future by entering:

```
rx Window.rexx
```

Naming AREXX Programs

AREXX programs can be named anything, but adopting a simple naming convention will make program management easier. Programs run from the Shell should have the .rexx extension to distinguish them from files run from other applications. Note that entering "rx Window" is equal to entering "rx Window.rexx."

Where Do AREXX Programs Go?

AREXX programs are usually stored in rexx:, but any directory can be used. The above program is stored in rexx: because you can run it without typing a path, and you will have all of your AREXX programs stored together. In addition, most applications search rexx: for AREXX macros.

AREXX System Files

The AREXX system files are provided on floppy disk and hard disk (if applicable). AREXX files are composed of shared libraries, the resident program (interpreter), and command utilities. Listed below is each file and its location on the Workbench disk.

SYS:LIBS Directory

rexsyslib.library
rexsupport.library
mathieedoubbas.library

SYSTEM: Directory

RexxMast

SYS:Rexxc Directory

HI
RXC
RXSET
TCO
TS
RX
RXLIB
TCC
TE
WaitForPort

Rexx:

This is a name which has been assigned to the directory which will be used to store your AREXX programs. This is where Example #1, Window.rexx, was stored.

Language features

Highlights of the AREXX programming language are:

- Command interface — AREXX provides you with another method of communication with programs. Through the command interface, AREXX can issue commands to any software that contains an AREXX port; this allows modification of its use and capabilities.
- Control of AmigaDOS — AREXX can be used to talk with and manipulate AmigaDOS. AREXX can replace many AmigaDOS scripts and in many instances is more powerful than AmigaDOS.

- **Tracing and Debugging** — AREXX has source-level debugging facilities that allow you to view a program step-by-step, thus reducing time required to develop and test programs. Tracing (or the internal interrupt system) permits the special handling of errors that would normally cause the program to prematurely abort.
- **Interpreted Execution** — The read-and-execute ability of AREXX skips the extra step of compiling a program. This simplifies learning and prototyping.
- **Automatic Resource Management** — AREXX automatically handles internal memory allocation related to the creation and removal of strings and other data.
- **Typeless Data** — Data is treated as individual character strings and variables do not have to be declared prior to use.
- **Function Libraries** — External function libraries are used to extend the capabilities of the language, or used as a bridge to other programs.

Program Examples

Before introducing the structure and syntax of the language, let's look at a few AREXX sample programs. Readers familiar with other high-level programming languages should find many points of similarity between AREXX and other languages. In the examples that follow, new terms are highlighted in the text as they are introduced, and will be covered in depth in the next few sections.

These short programs can be created using any text editor (like ED, MEmacs, etc.) or a word processor and then run from the Shell. If you use a word processor remember to save your program as a text (ASCII) file.

We'll begin with a program that displays a message on the console screen. Enter the following program in your text editor.

Sample 1. Topic

```
/* A simple program */  
say 'Amiga, The Computer For the Creative Mind.'
```

Save this program as "Rexx:Amiga.rexx".

Now, to run the program, go to your Shell and enter:

```
rx amiga
```

Remember that even though the full path and program name is "Rexx:Amiga.rexx" you don't need to type the "rexx:" prefix or the .rexx extension if the program has been saved in REXX:.

You should see the following in your Shell window:

```
Amiga, The Computer For the Creative Mind.
```

This program consists of a *comment line* that describes the program and an *instruction* that displays text on the console.

AREXX programs must begin with a comment line.

The initial `/*` says "I'm an AREXX program" to the interpreter when it searches for a program (or conversely, without the `/*` AREXX will not view it as an AREXX program). So remember to *always begin your programs with a comment line*. AREXX basically ignores comment lines when it executes a program. However, comment lines are extremely useful to you and others who may be reading your programs. When used wisely, they help make sense out of your program.

Instructions

Instructions are language statements that denote a certain action to be performed and always start with a *symbol*, in this case the word **say**. Symbols are translated to uppercase when the program is run, so the symbol **say** here is equivalent to **SAY**. Following **say** is an example of a *string*, which is a series of characters surrounded by quotes (''). Double quotes (") could also have been used to define the string.

In the next program we'll display a prompt for input and then read some information from the user. Each line (1 through 4) is numbered for reference. **Do not enter these reference numbers.**

Sample 2. Function

1. /* Calculate age in days */
2. say 'Please enter your age':
3. pull age
4. say 'You are about' age*365 'days old'

Save this program as REXX:Age.rexx and run it with the command

```
rx age
```

The program begins with a comment line (Line 1) that describes what the program will do.

Line 2 uses the **SAY** instruction to display a request for input on the console. Line 3 uses the **PULL** instruction to read a line of input from the user, which in this case is the user's age. The **PULL** instruction in line 3 takes the input, converts it to upper case letters (if necessary), and stores it in a *variable*. Variables are symbols which may be assigned a value. You may arbitrarily choose a variable name (see pages 10-27 and 10-28 for full details), but it always helps to choose a descriptive

name. Line 3 uses the name *age* to hold the number which you entered. Line 4 multiplies the variable **age** by 365 (the approximate number of days in a year) and issues the **SAY** instruction to display the result.

Note that the variable **age** did not have to be declared as a number, because its value was checked when it was actually used in the expression (this is what is meant by *typeless data*). To see what would happen if *age* wasn't a number, try rerunning the program with a non-numeric entry for the age. The resulting error message shows the line number and type of error that occurred, after which the program ends.

Sample 3. Do

The next program introduces the **do** instruction, which allows program statements to be executed repeatedly. It also illustrates the *exponentiation* operator, which is used to raise a number to an integral power. Remember that the first line of your program should be a comment line. Enter the program in your text editor or word processor and save it as REXX:calc.rexx. Then use the command **rx calc** to run the program.

```
/* Calculate some squares and cubes          */
do i = 1 to 10                                /* 10 iterations */
    say i i**2 i**3                          /* calculations */
end                                            /* end of loop */
say 'all done'
```

The **do** instruction causes all the statements between the **do** and **end** instructions to be executed 10 times. The variable **i** is the *index variable* for the loop and is incremented by 1 for each iteration (repetition). The number following the symbol **to** is the *limit* for the **do** instruction and could have been a variable or a full expression rather than just the constant 10.

Note that the statements within the loop have been indented. This is not required by the language, but it makes the program more readable, because you can easily visualize where the loop starts and stops. Also notice that you can include a comment on any line of your program as long as the comment begins with the symbol `/*` and ends with the symbol `*/`, which tells the AREXX interpreter that the characters contained within those symbols are to be ignored. Although AREXX ignores a comment, the comment is quite useful in explaining what a line in a given program is doing. As noted above, it's also good practice to include descriptive comments in your programs; it helps you and others better understand the program.

Sample 4. If

The `if` instruction allows statements to be conditionally executed. The numbers from 1 to 10 are classified as even or odd by dividing them by 2 and then checking the remainder.

```
/* Even or odd? */  
do i = 1 to 10          /* Begin loop — 10 iterations */  
  if i//2 = 0 then type = 'even'  
    else type = 'odd'  
  say i 'is' type  
end                      /* End loop */
```

This example introduces the `//` arithmetic operator, which calculates the remainder after a division operation. The `if` instruction tests whether the remainder is 0 and if the test turns out to be true, executes the **then** branch, thereby setting the variable **type** to even. If the remainder was not 0, the statement is false and the program will skip over the **then** branch and execute the **else** branch, thereby setting the variable **type** to odd.

Notice that the variable **type** is being assigned not to a number but to a string (either even or odd).

Sample 5. Function

The next example introduces the concept of a *function*, a group of statements that can be executed by mentioning the function name in a suitable context. Functions allow you to build large complex programs from smaller modules. Functions also permit the same code for similar operations in a different program.

Functions are specified in an expression as a name followed by an open parenthesis. One or more expressions called *arguments* may follow the parenthesis. These arguments pass information to the function for processing. Save this program as rexx:square.rexx and run the program using the **rx** command.

```
/* Defining and calling a function */
do i = 1 to 5
    say i square(i)          /* call the "square" function */
end
exit                        /* all done */
square:                     /* function name */
    arg x                   /* get the argument */
    return x**2             /* square it and return */
```

The function **square** is defined in the lines following the *label square*: up through the entire **return** instruction. It is here where the actual squaring of the values will be calculated.

Two new instructions are introduced here: **arg** retrieves the value of the argument string (in this case "i") and **return** passes the function's result back to the point where the function was called.

Let's follow how this program executes.

- A loop is set up (**do, end**) with an index variable "i" that will increment by 1. The loop will iterate (repeat) five times.

- In this loop is an expression that calls a function when the expression is evaluated and displays the function's result using the **say** instruction.
- Once the function is called, the program looks for the function label (**square:**), then it retrieves the argument, in this case "i", performs a calculation (squares it) and returns to the place where the function was called (the line within the **do/end** loop).
- The **exit** instruction will end the program after the program finishes its last loop.

Sample 6. Trace

The next instruction will help you check your program for errors. This new instruction called **trace** activates a tracing feature of AREXX.

```
/* Demonstrate "results" tracing */
trace results
sum = 0 ; sumsq = 0;
do i = 1 to 5
    sum = sum + i
    sumsq = sumsq + i**2
end
say 'sum = ' sum 'sumsq = ' sumsq
```

When you run this program, the console displays the source lines as they are executed and shows the final results of expressions. The result is `sum = 15` and `sumsq = 55`. This makes it easy to tell what the program is really doing, and helps reduce the time required to develop and test a new program.

One minor point is illustrated in the above program: the third line shows two distinct statements separated by a semicolon (;). The semicolon is an example of a *special character*, characters that have particular meaning within AREXX programs.

Sample 7. Grades

Enter and study the following program, which calculates the final grade for a given student. It prompts the user for 4 essay grades, a class participation grade, and then calculates the final grade based on the following percentages:

The average of Essay 1 and Essay 2 is worth 30% of the final grade. The average of Essay 3 and Essay 4 is worth 45% of the final grade. Participation is worth 25% of the final grade.

```
/* grading program */
say "Hello, I will calculate your grades for you."
response = 0
do while response = "Q" /* loop while response isn't 'Q' */
    say "Please enter all grades for the student"
    say "Essay 1:"
    pull es1
    say "Essay 2:"
    pull es2
    say "Essay 3:"
    pull es3
    say "Essay 4:"
    pull es4
    say "Participation:"
    pull p
    Final = (((es1 + es2)/2)*.3) + (((es3 + es4)/2)*.45) + (p*.25)
    say "Your final grade for this student is . . ." Final
    say "Would you like to continue? (Q for quit)"
    pull response
end
exit
```

NOTE: "Pull" in the third to last line of the program will return uppercase letters.

After the program displays the final grade it asks the user if he would like to continue. The response is pulled and if it does not equal Q (quit), the loop continues. If the response equals Q, then the program quits the loop and exits. This way, you don't have to run the program again and again for each student. The program is only invoked once and you can continue until all the students' final grades have been calculated.

The following sections will present further information on the language statements illustrated here and will introduce others that have not yet been shown. AREXX is a relatively small language and there are relatively few words and rules to learn.

Review and Additional Notes

- To manually start the AREXX server, double-click on the RexxMast icon or type the **RexxMast** command at a Shell prompt. Usually RexxMast will start from the hard disk when the Amiga is first turned on.
- To run AREXX programs, type the command **rx** followed by the program name (e.g., **rx test** will run the program `rexx:test.rexx`).
- Short programs (i.e., a one-line program) can be entered directly at the command line by enclosing the program line in double quotes. For example, the following program will print five files named `myfile.1` through `myfile.5`

```
rx "do i = 1 to 5; address command 'copy my file.' ||i 'prt:.'; end"
```

- When using a program that interfaces with AREXX, you can run AREXX programs from within that application by using either the menu or command options of that program. Refer to the application's documentation for more information.

- The REXX server can be closed by issuing the **rx** command at the Shell prompt. This is usually unnecessary as AREXX consumes little memory.
- You can declare a system-wide directory for frequently used AREXX programs by assigning the name REXX: to the appropriate directory. For example, **assign rexx: DF0:rexx**.
- For advanced users: The REXX server also supports an additional host port (named AREXX) for launching asynchronous commands. A command sent to the AREXX port will return immediately without indicating an error, making it roughly the same as "run rx <program>", but without the overhead of an extra Shell process.
- **Workbench Execution.** The **rx** command can be launched from either a tool or a project icon. It accepts tooltype arguments using CONSOLE for a window specification and CMD for a command string. If no CMD string is supplied, the command will attempt to execute the (project) file as an AREXX program. The rx command will also attempt to start REXXMaster if it's not active.

For example:

Icon type: Project

Default Tool: sys:rexxc/rx

Possible Tool Types:

Console = con:0/0/640/200/Example/Close

CMD = rexxprogram arg1 arg2

- **Rxlib Command Utility.** The **rxlib** command opens a function library so that it can be called from AREXX and can also be used to list the currently-defined functions. The command syntax is:

rxlib library-name priority offset version

The priority argument gives the search priority and must be in the range -100 to 100 inclusive. The offset argument is the actual integer offset to the library's entry point and should be documented with each library. The version is the required library version and can usually be omitted, as the default is to load any version.

Caution: Calling a library with the incorrect entry point may crash the system. Anyone developing an external function library should carefully document the offset to the query (look-up) entry.

For example:

```
rxlib rexxsupport.library 0 -30 0
```

The rexxsyslib. library is automatically opened.

Elements of AREXX

This section introduces the rules and concepts that make up the AREXX language. The intent here is to convey a practical understanding of how AREXX's elements fit together to form programs rather than to present an overly formal language definition.

Format

AREXX programs are composed of ASCII characters and may be created using any text editor. No special formatting of the program statements is required or imposed on the programmer.

NOTE: AREXX supports the extended ASCII character set (æ, å, etc.). These extended characters are recognized as ordinary printing characters and will be correctly mapped from lowercase to uppercase. However, they are not considered as AREXX symbol characters.

Tokens

Tokens are the smallest distinct entities or words of the language. A token may be a series of characters, as in the symbol **MyName**, or just a single character like the "+" operator. Tokens can be categorized as follows:

- *comments*
- *symbols*
- *strings*
- *operators*
- *special characters*

Comment Tokens. Any group of characters beginning with the sequence `"/"` and ending with `"*/"` defines a *comment* token. Comments may be placed anywhere in a program and cost little in terms of execution speed, since the interpreter ignores them when it scans the program. Comments may be "nested" within one another, but each `"/"` must have a matching `"*/"` in the program. For example:

```
/* Your basic comment */  
/* a /* nested! */ comment */
```

Each AREXX program must begin with a comment line.

Symbol Tokens. Any group of the characters a-z, A-Z, 0-9, and .!?\$_ defines a *symbol* token. Symbols are translated to uppercase as the interpreter scans the program, so the symbol MyName is equivalent to MYNAME. Four types of symbols are recognized:

- *Fixed* symbols begin with a digit (0-9) or a period (.).
- *Simple* symbols do not begin with a digit and do not contain any periods.
- *Stem* symbols have exactly one period at the end of the symbol name.
- *Compound* symbols include one or more periods in the interior of the name.

Stems and compound symbols have special properties that make them useful for building arrays and lists.

Symbol Values. The value used for a fixed symbol is always the symbol name itself (as translated to uppercase). Simple, stem, and compound symbols are called *variables* and may be assigned a value during the course of the program execution. A variable is *uninitialized* if it has not yet been assigned a value; the value used for an uninitialized variable is just the variable name itself.

For example:

123.45	/* a fixed symbol */
MyName	/* a simple symbol */
a.	/* a stem symbol */
a.1.Index	/* a compound symbol */

String Tokens. A group of characters beginning and ending with a quote (') or double quote (") delimiter defines a *string* token. The delimiter character itself may be included within the string by a double-delimiter sequence ('' or '').

The number of characters in the string is called its length, and a string of length zero is called a *null string*. A string is treated as a *literal* in an expression; its value is just the string itself.

Strings followed immediately by an "X" or "B" character that is not part of a longer symbol are classified as *hex* or *binary* strings, respectively, and must be composed of hexadecimal digits (0-9, A-F) or binary digits (0,1). Blanks are permitted at byte boundaries for added readability. Hex and binary strings are convenient for specifying non-ASCII characters and for machine-specific information like addresses in a program. They are converted immediately to the packed (machine compressed) internal form.

For example:

"Now is the time"	/* a simple example */
""	/* a null string */
'Can''t you see??'	/* Can't you see?? */
'4A 3B C0'X	/* a hex string */
'00110111'b	/* binary for the character '7' */

Operators. The characters ~ + - * / = > < & | ^ may be combined in the sequences shown in the following table to form *operator* tokens. Operator sequences may include leading, trailing and embedded blanks, all of which are removed when the program is scanned. In addition to the above characters, the *blank* character is treated as a concatenation operator or special character.

Each operator has an associated priority that determines the order in which operations will be performed in an expression. Operators with higher priorities (8) are performed before those with lower priorities (1).

Operator Sequences		
Sequence	Priority	Operator Definition
~	8	Logical NOT
+	8	Prefix Conversion
-	8	Prefix Negation
**	7	Exponentiation
*	6	Multiplication
/	6	Division
%	6	Integer Division
//	6	Remainder
+	5	Addition
-	5	Subtraction
	4	Concatenation
(blank)	4	Blank Concatenation
= =	3	Exact Equality
~ = =	3	Exact Inequality
=	3	Equality
~ =	3	Inequality
>	3	Greater Than
> = or ~ <	3	Greater Than or Equal To
<	3	Less Than
< = or ~ >	3	Less Than or Equal To
&	2	Logical AND
	1	Logical Inclusive OR
^ or &&	1	Logical Exclusive OR

Special Character Tokens. The characters :(),, are each treated as a separate *special character* token and have particular meanings within an AREXX program. Blanks adjacent to these special characters are removed, except for those preceding an open parenthesis or following a closed parenthesis.

Colon (:) A colon, if preceded by a symbol token, defines a *label* within the program. *Labels* are locations in the program to which control may be transferred under various conditions.

Opening and Closing Parentheses (()) Parentheses are used in expressions to group operators and operands into subexpressions in order to override the normal operator priorities. An open parenthesis also serves to identify a *function call* within an expression. A symbol or string followed immediately by an open parenthesis defines a function name. Parentheses must always be balanced within a statement.

Semicolon (;) The semicolon acts as a program statement terminator. Several statements may be placed on a single source line if separated by semicolons.

Comma (,) A comma token acts as the continuation character for statements that must be entered on several source lines. It is also used to separate the argument expressions in a function call.

Clauses

Tokens are grouped together to form *clauses*, the smallest language unit that can be executed as a statement. Each AREXX clause can be classified as follows:

- *null*
- *label*
- *assignment*
- *instruction*
- *command*

The classification process is very simple, since no more than two tokens are required to classify any clause. Assignment, instruction, and command clauses are *statements*.

Null Clauses. Lines consisting only of blanks or comments are called *null* clauses. They have no function in the execution of a program, except to aid its readability and to increment the source line count. Null clauses may appear anywhere in a program.

For example:

```
/* perform annuity calculations */
```

This comment line is a null clause.

Label Clauses. A symbol followed immediately by a colon (:) defines a *label* clause. A label acts as a placemaker in the program, but no action occurs with the execution of a label. The colon is considered as an implicit clause terminator, so each label stands as a separate clause. Label clauses may appear anywhere in a program.

For example:

```
start:                                /* begin execution */  
syntax:                              /* error processing */
```

The above lines end with a colon, signifying a specific place in the program.

Assignment Clauses. Assignments are identified by a variable symbol followed by an "=" operator. In this context *the "=" operator's normal definition (an equality comparison) is overridden and it becomes an assignment operator.* The tokens to the right of the "=" are evaluated as an expression and the result is *assigned* to (becomes the value of) the variable symbol.

For example:

```
when = 'Now is the time'  
answ = 3.14 * fact(5)
```

In the above lines the equality comparison definition is superseded and the equal sign (=) instead assigns the value

'Now is the time' to the variable 'when', and assigns the result of $3.14 * \text{fact}(5)$ to the variable 'answ'.

Instruction Clauses. Instructions begin with certain keyword symbols, each of which denote a particular action to be performed. Instruction keywords are recognized only at the beginning of a clause and may otherwise be used freely as symbols (although such use may become confusing at times).

For example:

```
drop a b c           /* reset variables */
say 'please'         /* a polite program */
if j > 5 then leave;  /* several instructions */
```

Command Clauses. Commands are any AREXX expression that can't be classified as one of the preceding types of clauses. The expression is evaluated and the result is issued as a command to an external *host*, which might be the native operating system (like AmigaDOS) or an application program.

For example:

```
'delete' 'myfile'      /* a DOS command */
'jump' current + 10    /* an editor command? */
```

In the first example the command 'delete' is not recognized as an AREXX command; it is sent to the external host, in this case AmigaDOS which does understand the command. Similarly, the 'jump' command in the second example is supposedly understood by the external host — in this case a text editor. See also **ADDRESS ()**.

Multiple Clauses

Several clauses can be placed on a single line by separating them with semicolons (;).

Clause Classification

The process by which program lines are divided into clauses and then classified is important in understanding the operation of an AREXX program. The language interpreter splits the program source into groups of clauses as the program is read, using the end of each line as a clause separator and applying the continuation rule as required. These groups of one or more clauses are then tokenized and each clause is classified into one of the above types. Note that seemingly small syntactic differences may completely change the semantic content of a statement. For example,

```
SAY 'Hello, Bill'
```

is an instruction clause and will display "Hello, Bill" on the console, but

```
'SAY 'Hello, Bill'
```

is a command clause, and will issue "SAY Hello, Bill" as a command to an external program. The presence of the leading null string changes the classification from an instruction clause to a command clause.

Clause Continuation

The end of a source line normally acts as the implicit end of a clause. A clause can be continued on the next source line by ending the line with a comma (see Multiple Clauses). The comma is ignored by the program, and the next line is considered as a continuation of the clause. There is no limit to the number of continuations that may occur (except for those limits imposed by the command buffer). String and comment tokens are automatically continued if a line ends before the closing delimiter has been found, and the newline (i.e., enter) character is not considered to be part of the token.

Expressions

Expression evaluation is an important part of AREXX programs since most statements include at least one expression.

Expressions are composed of:

- *Strings*. Used as literals (the values as shown in uppercase) in an expression; their value in an operation is just the string itself.
- *Symbols*. Fixed symbols are also literals (remember that symbols are always translated to uppercase), but variable symbols may have an assigned value.
- *Operators*. Operator tokens represent the predefined operations of AREXX; each operator has an associated priority that determines the order in which operations will be performed.
- *Parentheses*. Parentheses may be used to alter the normal order of evaluation in the expression or to identify *function* calls. A symbol or string followed immediately by an open parenthesis define the function name, and the tokens between the opening and (final) closing parenthesis form the *argument list* for the function.

For example, the expression "**J 'factorial is' fact(J)**" is composed of a symbol **J**, a blank operator, the string **'factorial is'**, another blank, the symbol **fact**, an open parenthesis, the symbol **J** again, and a closing parenthesis. **FACT** is a function name and **(J)** is its argument list, in this case the single expression **J**.

Symbol Resolution

Before the evaluation of an expression proceeds, the interpreter must obtain a value for each symbol in the expression. For fixed symbols the value is just the symbol name itself, but variable symbols must be looked up in the current symbol table. In the example above, the expression after symbol resolution would be "**3 'factorial is' FACT(3)**" assuming that the symbol **J** had the value 3.

Suppose that the previous example above had been "**FACT(J) 'is' J 'factorial'.**" Would the second occurrence of symbol **J** still resolve to 3 in this case? In general, function calls may have "side effects" that include altering the values of variables, so the value of **J** might have been changed by the call to **FACT**. In order to avoid ambiguities in the values assigned to symbols during the resolution process, AREXX guarantees a strict *left-to-right resolution order*. Symbol resolution proceeds irrespective of operator priority or parenthetical grouping. If a function call is found, the resolution is suspended while the function is evaluated. Note that it is possible for the same symbol to have more than one value in an expression.

Order of Evaluation

After all symbol values have been resolved the expression is evaluated based on operator priority and subexpression grouping. Operators of higher priority are evaluated first. AREXX does not guarantee an order of evaluation among

operators of equal priority, and does not employ a "fast path" evaluation of boolean operations. For example, in the expression

`(1 = 2) & (FACT(3) = 6)`

the call to the **FACT** function will be made since the first term of the AND (&) operation is 0. This example points out that AREXX will continue reading left to right, even though the given example is false and will return a value of 0.

Numbers and Numeric Precision

An important class of operands are those representing numbers. Numbers consist of the characters **0-9**, **.**, **+**, **-**, and blanks; **e** or **E** may follow a number to indicate *exponential notation*, in which case it must be followed by a (signed) integer.

Both string tokens and symbol tokens may be used to specify numbers. Since the language is typeless, variables do not have to be declared as "numeric" before use in an arithmetic operation. Instead, each value string is examined when it is used to verify that it represents a number. The following examples are all valid numbers:

33
" 12.3 "
0.321e12
' + 15. '

Note that leading and trailing blanks are permitted and that blanks may be embedded between a "+" or "-" sign and the number body (but not within the body).

Boolean Values

The numbers 0 and 1 are used to represent the boolean values **False** and **True**, respectively. The use of a value other than 0 or 1 when a boolean operand is expected will generate an error. Any number equivalent to 0 or 1, for example "0.000" or "0.1E1", is also acceptable as a boolean value.

Numeric Precision

AREXX allows the basic precision used for arithmetic calculations to be modified while a program is executing. The number of significant figures used in arithmetic operations is determined by the Numeric Digits setting and may be modified using the **NUMERIC** instruction.

The number of decimal places used for a result depends on the operation performed and the number of decimal places in the operands. Unlike many languages, AREXX preserves trailing zeroes to indicate the precision of the result. If the total number of digits required to express a value exceeds the current Numeric Digits setting, the number is formatted in *exponential notation*. Two such formats are provided:

- In **SCIENTIFIC** notation, the exponent is adjusted so that a single digit is placed to the left of the decimal point.
- In **ENGINEERING** notation, the number is scaled so that the exponent is a multiple of 3 and the digits to the left of the decimal point range from 1 to 999.

The numeric precision and format can be set using the NUMERIC instruction (See page 10-61).

Operators

Operators can be grouped into four categories:

- *Arithmetic* operators require one or two numeric operands and produce a numeric result.
- *Concatenation* operators join two strings into a single string.
- *Comparison* operators require two operands and produce a boolean (0 or 1) result.
- *Logical* operators require one or two boolean operands and produce a boolean result.

Arithmetic Operators. The arithmetic operators are listed in the Table below. Note the inclusion of the integer division (%) and remainder (//) operators, along with the usual arithmetic operations. The result of an arithmetic operation is always formatted based on the current Numeric Digits setting and will never have leading or trailing blanks.

Arithmetic Operators		
Sequence	Priority	Operation
+	8	Prefix Conversion
-	8	Prefix Negation
**	7	Exponentiation
*	6	Multiplication
/	6	Division
%	6	Integer Division
//	6	Remainder
+	5	Addition
-	5	Subtraction

NOTE: The \rightarrow (arrow) used in program examples throughout this chapter is to be understood as "evaluates as." This \rightarrow (arrow) will not be displayed when a program is run. It is used for clarification purposes only. See examples below.

Prefix Conversion (+). This unary (the positive or negative sign on a number) operator converts the operand to an internal numeric form and formats the result based on the current Numeric Digits settings. This causes any leading and trailing blanks to be removed and may result in a loss of precision.

For example:

<code>+ ' 3.12 '</code>	\rightarrow 3.12
<code>+ 1.5001</code>	\rightarrow 1.500 /* If digits = 3 */

Prefix Negation (-). This unary operator negates the operand. The result is formatted based on the current Numeric Digits setting.

For example:

<code>- ' 3.12 '</code>	\rightarrow -3.12
<code>- 1.5E2</code>	\rightarrow -150

Exponentiation ().** The left operand is raised to the power specified by the right operand, which must be an integer. The number of decimal places for the result is the product of the exponent and the number of decimal places in the base.

For example:

$2^{**}3$	$\rightarrow 8$
$3^{**}-1$	$\rightarrow .33333333$
$0.5^{**}3$	$\rightarrow 0.125$

Multiplication (*). The product of two numbers is computed. The number of decimal places for the result is the sum of the decimal places of the operands.

For example:

$12 * 3$	$\rightarrow 36$
$1.5 * 1.50$	$\rightarrow 2.250$

Division (/). The quotient of two numbers is computed. The number of decimal places for the result depends on the current setting of Numeric Digits; the number is formatted to the maximum precision required.

For example:

$6 / 3$	$\rightarrow 2$
$8 / 3$	$\rightarrow 2.66666667$

Integer Division (%). The quotient of two numbers is computed, and the integer part of the quotient is used as the result.

For example:

$5 \% 3$	$\rightarrow 1$
$-8 \% 3$	$\rightarrow -2$

Remainder (//). The result is the remainder after the two operands are divided. The remainder for " a/b " is calculated as " $a - (a \% b) * b$ ". If both operands are positive integers, this operation yields the usual modulo result.

For example:

$5 // 3$	$\rightarrow 2$
$-5 // 3$	$\rightarrow -2$
$5.1 // 0.2$	$\rightarrow 0.1$

Addition (+). The sum of two numbers is computed. The number of decimal places for the result is the larger of the decimal places of the operands.

For example:

$$\begin{array}{rcl} 12 + 3 & \rightarrow & 15 \\ 3.1 + 4.05 & \rightarrow & 7.15 \end{array}$$

Subtraction (-). The difference of two numbers is computed. As in the case of addition, the number of decimal places for the result is the larger of the decimal places of the operands.

For example:

$$\begin{array}{rcl} 12 - 3 & \rightarrow & 9 \\ 5.55 - 1.55 & \rightarrow & 4.00 \end{array}$$

Concatenation Operators. AREXX defines two concatenation operators, both of which require two operands. The first, identified by the operator sequence " || ", joins two strings into a single string with no intervening blank. The second concatenation operation is identified by the blank operator and joins the two operand strings with one intervening blank.

An implicit concatenation operator is recognized when a symbol and a string are directly abutted in an expression. Concatenation by abuttal uses the " || " operator, and behaves exactly as though the operator had been provided explicitly.

For example:

'why me,' 'Mom?'	→ why me,Mom?
'good' 'times'	→ good times
one'two'three	→ ONetwoTHREE

Comparison Operators. Comparisons are performed in one of three modes and always result in a boolean value (0 or 1).

- *Exact* comparisons proceed character-by-character, including any leading blanks that may be present.

- *String* comparisons ignore leading blanks and add blanks to the shorter string if necessary.
- *Numeric* comparisons first convert the operands to an internal numeric form using the current Numeric Digits setting and then perform a standard arithmetic comparison.

Except for the exact equality and exact inequality operators, all comparison operators dynamically determine whether a string or numeric comparison is to be performed. A numeric comparison is performed if both operands are valid numbers. Otherwise, the operands are compared as strings.

Comparison Operators			
Sequence	Priority	Operation	Mode
<code>==</code>	3	Exact Equality	Exact
<code>~=</code>	3	Exact Inequality	Exact
<code>=</code>	3	Equality	String/Numeric
<code>~=</code>	3	Inequality	String/Numeric
<code>></code>	3	Greater Than	String/Numeric
<code>>=</code> or <code>~<</code>	3	Greater Than or Equal	String/Numeric
<code><</code>	3	Less Than	String/Numeric
<code><=</code> or <code>~></code>	3	Less Than or Equal	String/Numeric

Logical (Boolean) Operators. AREXX defines the four logical operations NOT, AND, OR, and Exclusive OR, all of which require boolean operands and produce a boolean result. (Refer to the Glossary at the back of this manual for an explanation of boolean.) Boolean operands must have values of either 0 (**False**) or 1 (**True**). An attempt to perform a logical operation on a non-boolean operand will generate an error.

Logical Operators		
Sequence	Priority	Operation
~	8	NOT (Inversion)
&	2	AND
	1	OR
^ or &&	1	Exclusive OR

Stems and Compound Symbols

Stems and compound symbols have special properties that allow for some interesting and unusual programming. A compound symbol can be regarded as having the structure **stem**. $n_1.n_2.n_3 \dots n_k$ where the leading name is a stem symbol and each node $n_1 \dots n_k$ is a fixed or simple symbol. Whenever a compound symbol appears in a program, its name is *expanded* by replacing each node with its current value as a (simple) symbol. The value string may consist of any characters, including embedded blanks, and is not converted to uppercase. The result of the expansion is a new name that is used in place of the compound symbol. For example, if **J** has the value **3** and **K** has the value **7**, then the compound symbol **A.J.K** will expand to **A.3.7**.

Stem symbols provide a way to initialize a whole class of compound symbols. When an assignment is made to a stem symbol, it assigns that value to all possible compound symbols derived from the stem. Thus, the value of a compound symbol depends on the prior assignments made to itself or its associated stem.

Compound symbols can be regarded as a form of associative or content-addressable memory. For example, suppose that you needed to store and retrieve a set of names and telephone numbers. The conventional approach would be to set up two arrays **NAME** and **NUMBER**, each indexed by an integer running from one to the number of entries. A number would be looked up by scanning the name array until the given name was found, say in **NAME.12**, and then retrieving **NUMBER.12**. With compound symbols, the symbol **NAME** could hold the name to be looked-up, and **NUMBER.NAME** would then expand to **NUMBER.CBM** (for example), which would be the corresponding number.

Of course, compound symbols can also be used as conventional indexed arrays, with the added convenience that only a single assignment (to the stem) is required to initialize the entire array.

Enter the following phonebook program and save it as "rexx:phone.rexx".

```
/* A simple telephone phone book to */  
/* demonstrate compound variables */
```

```
if arg() ~ = 1 then do  
  say 'USAGE: rx phone name'  
  exit 5  
end
```

```
/* open window to display phone numbers/addresses in */
```

```
call open out, "con:0/0/640/60/AREXX Phonebook"
```

```
if ~ result then do  
  say "Open failure . . . sorry"  
  exit 10  
end
```



```
/* number definitions */
number.      = '(not found)'
number.wsh   = '(555) 001-0001'
addr.        = '(not found)'
number.CBM   = '(555) 002-0002'
addr.CBM     = '1200 Wilson Dr., West Chester, PA 19380'

/* (work is done here) */

arg name     /* the name */

    call writeln out,name || "'s number is" number.name
    call writeln out,name || "'s address is" addr.name
    call writeln out,"Press RETURN to exit"
    call readln out
exit
```

NOTE: If you are new to AREXX you may wish to skip ahead to Instructions on page 10-50. The information beginning with The Execution Environment and ending with Resource Tracking on page 10-49 is intended for advanced Amiga users. This information assumes a working knowledge of the Amiga operating system and familiarity with the ROM Kernel Manuals published by Addison Wesley.

The Execution Environment

The AREXX interpreter provides a uniform execution environment by running each program as a separate process in the Amiga's multitasking operating system. This allows for a flexible interface between an external host program and the interpreter, as the host can either proceed concurrently with its operations or can simply wait for the interpreted program to finish.

The External Environment

The external environment of a program includes its process structure, input and output streams, and current directory. When each AREXX process is created, it inherits the input and output streams and current directory from its *client*, the external program that invoked the AREXX program. The current directory is used as the starting point in a search for a program or data file.

External Programs. The external environment usually includes one or more external programs with which the AREXX program may communicate. Any program that supports a suitable interface (AREXX port) can receive commands from AREXX programs.

The Internal Environment

The internal environment of an AREXX program consists of a static global structure and one or more storage environments. The global data values are fixed at the time the program is invoked and include the argument strings, program source code, and static data strings. The storage environment includes the symbol table used for variable values, the numeric options, trace options, and host address strings. While the global environment is unique, there may be many storage environments during the course of the program execution. Each time an internal function is called a new storage environment is activated and initialized. The initial values for most fields are inherited from the previous environment, but values may be changed afterwards without affecting the caller's environment. The new environment persists until control returns from the function.

Argument Strings. A program may receive one or more argument strings when it is first invoked. These arguments persist for the duration of the program and are not altered. The number of arguments a program receives depends in part on the mode of invocation. AREXX programs invoked as commands normally have only one argument string, although the "command tokenization" option may provide more than one. A program invoked as a function can have any number of arguments if called as an internal function, but external functions are limited to a maximum of 15 arguments.

The argument strings can be retrieved using either the **ARG** instruction or the **ARG()** Built-In function. **ARG()** can also return the total number of arguments, or the status (as "exists" or "omitted") of a particular argument.

The Symbol Table. Every storage environment includes a *symbol table* to store the value strings that have been assigned to variables. This symbol table is organized as a two-level *binary tree*, a data structure that provides an efficient look-up mechanism. The primary level stores entries for simple and stem symbols and the secondary level is used for compound symbols. All of the compound symbols associated with a particular stem are stored in one tree, with the root of the tree held by the entry for the stem.

Symbols are not entered into the table until an assignment is made to the symbol. Once created, entries at the primary level are never removed, even if the symbol subsequently becomes uninitialized. Secondary trees are released whenever an assignment is made to the stem associated with the tree.

Input and Output

Most computer programs require some means of communicating with the outside world, either to accept input data or to pass along results. The REXX language includes only a minimal specification of input and output (I/O) operations, leaving the choice of additional functionality to the language implementor. This is in keeping with the design of many computer languages. For instance, the C language has no statements dedicated to I/O, but instead relies on a standardized set of I/O functions.

AREXX extends the I/O facilities of REXX by providing Built-In functions to manipulate external files. Files are referenced by a *logical name* associated with the file when it is first opened. The initial input and output streams are given the names **STDIN** (Standard Input) and **STDOUT** (Standard Output).

AREXX maintains a list of all of the files opened by a program and automatically closes them when the program finishes. There is no theoretical limit to the number of files that may be open simultaneously, although there is a limit imposed by available memory.

Resource Tracking

AREXX provides complete tracking for all of the dynamically allocated resources that it uses to execute a program. These resources include memory space, DOS files, and related structure, and the message port structure supported by AREXX. The tracking system was designed to allow a program to shut down at any point (perhaps due to an execution error) without leaving any hanging resources.

It is possible to go outside of the interpreter's resource tracking net by making calls directly to the Amiga's operating system from within an AREXX program. In this case it is the programmer's responsibility to track and return any resources allocated outside of the AREXX resource tracking system. AREXX provides a special interrupt facility so that a program can retain control after an execution error, perform the required cleanup, and then make an orderly exit.

Instructions

Instruction clauses are identified by an initial keyword symbol that is not followed by a colon (:) or an equals (=) operator. Each instruction signifies a specific action and may be followed by one or more subkeywords, expressions, or other instruction specific information. Instruction keywords and subkeywords are recognized only in this specific context and are therefore not "reserved words" in the usual sense of the term. Keywords may be used freely as variables or function names, although such usage may become confusing at times.

In the descriptions that follow, keywords are shown in uppercase and optional parts of the instruction are enclosed in brackets. Alternative selections are separated by a vertical bar (|) and required alternatives are enclosed in braces ({}).

ADDRESS

Usage: ADDRESS [*symbol* | *string* | [[*VALUE*] [*expression*]]

This instruction specifies a *host address* for commands issued by the interpreter. A host address is the name associated within an external program to which commands can be sent. External

hosts are described on page 10-75. AREXX maintains two host addresses: a current and a previous value. Whenever a new host address is supplied, the previous address is lost and the current address becomes the previous one. These host addresses are part of a program's storage environment and are preserved across internal function calls. The current address can be retrieved with the Built-In function **ADDRESS** (). There are four distinct forms for the **ADDRESS** instruction:

- **ADDRESS {string | symbol} expression.** The expression is evaluated and the result is issued to the host specified by the string or symbol, which is taken as a literal. No changes are made to the current or previous address strings. This provides a convenient way to issue a single command to an external host without disturbing the current host addresses. The return code from the command is treated as it would be from a command clause.
- **ADDRESS {string | symbol}.** The string or symbol, taken as a literal, specifies the new host address. The current host address becomes the previous address.
- **ADDRESS [VALUE] expression.** The result of the expression specifies the new host address and the current address becomes the previous address. The **VALUE** keyword may be omitted if the first token of the expression is not a symbol or string.
- **ADDRESS.** This form interchanges the current and previous hosts. Repeated execution will therefore "toggle" between the two host addresses.

For example:

address edit	/* set a new host address */
address edit 'top'	/* move to the top */
address VALUE edit in	/* compute a new host address */
address	/* swap current and previous */

```
do                                     /* begin block */
  if i>3 then break                   /* all done? */
  a = a + 1
  y.a = name
end                                   /* end block */
```

CALL

Usage: **CALL** {*symbol* | *string*} [*expression*] [,*expression*, ...]

The **CALL** instruction is used to invoke an internal or external function. The function name is specified by the symbol or string token, which is taken as a literal. Any expressions that follow are evaluated and become the arguments to the called function. The value returned by the function is assigned to the special variable **RESULT**. It is not an error if a result string is not returned. In this case the variable **RESULT** is **DROPPed** (becomes uninitialized).

The linkage to the function is established dynamically at the time of the call. AREXX follows a specific search order in attempting to locate the called function.

For example:

```
call center name,length + 4,' + '
```

DO

Usage: **DO** [[*var = exp*]/[*exp*] [TO *exp*] [BY *exp*]] [FOR *exp*]
[FOREVER] [WHILE *exp* | UNTIL *exp*]

The **DO** instruction begins a group of instructions to be executed as a block. The range of the **DO** instruction includes all statements up to and including an eventual **END** instruction. There are two basic forms of the instruction:

- The **DO** keyword by itself defines a block of instructions to be executed once.
- If any iteration specifiers follow the **DO** keyword, the block of instructions is executed repeatedly until a termination condition occurs.

An iterative **DO** instruction is sometimes called a *loop*, since the interpreter “loops back” to perform the instruction repeatedly. The various parts of the **DO** instruction are described below.

Initializer expression. An initializer expression of the form "**variable = expression**" defines the *index variable* of the loop. The expression is evaluated when the **DO** range is first activated and the result is assigned to the index variable. On subsequent iterations an expression of the form "**variable = variable + increment**" is evaluated, where the increment is the result of the **BY** expression. If specified, the initializer expression must precede any of the other subkeywords.

BY expression. The expression following a **BY** symbol defines the increment to be added to the index variable in each subsequent iteration. The expression must yield a numeric result, which may be positive or negative and need not be an integer. The default increment is 1.

TO expression. The result of the **TO** expression specifies the upper (or lower) limit for the index variable. At each iteration the index variable is compared to the **TO** result. If the increment (**BY** result) is positive and the variable is greater than the limit, the **DO** instruction terminates and control passes to the statement following the **END** instruction. The loop terminates if the increment is negative and the index variable is less than the limit.

FOR expression. The **FOR** expression must yield a positive whole number when evaluated and specifies the maximum number of iterations to be performed. The loop terminates when this limit is reached irrespective of the value of the index variable.

FOREVER. The **FOREVER** keyword can be used if an iterative **DO** instruction is required but no index variable is necessary. The loop will be terminated by a **LEAVE** or **BREAK** instruction contained within the loop body.

WHILE expression. The **WHILE** expression is evaluated at the beginning of each iteration and must result in a boolean value. The iteration proceeds if the result is 1 (or true); otherwise, the loop terminates.

UNTIL expression. The UNTIL expression is evaluated at the end of each iteration and must result in a boolean value. The instruction continues with the next iteration if the result is 0 (or false), and terminates otherwise.

The initializer **BY**, **TO** and **FOR** expressions are evaluated only when the instruction is first activated, so the increment and limits are fixed throughout the execution. Note that a limit need not be supplied. For example, the instruction "**DO i=1**" will simply count away forever. Note also that only one of the **WHILE** or **UNTIL** keywords can be specified.

For example:

```
/* Examples of DO */
limit = 20; number = 1
do i = 1 to limit for 10 while number < 20
    number = i*number
    say "Iteration" i "number=" number
end

number = number/3.345; i = 0
do number for limit/5
    i = i + 1
    say "Iteration" i "number=" number
end
```

This is the output:

```
Iteration 1 number = 1
Iteration 2 number = 2
Iteration 3 number = 6
Iteration 4 number = 24
Iteration 1 number = 7.17488789
Iteration 2 number = 7.17488789
Iteration 3 number = 7.17488789
Iteration 4 number = 7.17488789
```

NOTE: If a FOR limit is also present, the initial expression is still evaluated, but the result need not be a positive integer.

For example:

```
do 5; say hi; end
```

DROP

Usage: DROP variable [variable . . .]

The specified variable symbols are reset to their uninitialized state, in which the value of the variable is the variable name itself. It is not an error to **DROP** a variable that is already uninitialized. **DROPPing** a stem symbol is equivalent to **DROPPing** the values of all possible compound symbols derived from that stem.

For example:

```
a = 123           /* assign a value */
drop a b          /* drop some      */
say a b           /* → A B */
```

ECHO

Usage: ECHO [expression]

The **ECHO** instruction is a synonym for the **SAY** instruction. It displays the expression result on the console.

For example:

```
echo "You don't SAY!"
```

ELSE

Usage: ELSE [;] [conditional statement]

The **ELSE** instruction provides the alternative conditional branch for an **IF** statement. It is valid only within the range of an **IF** instruction and must follow the conditional statement of the **THEN** branch. If the **THEN** branch wasn't executed, the statement following the **ELSE** clause is performed.

ELSE clauses always bind to the nearest (preceding) **IF** statement. It may be necessary to provide “dummy” **ELSE** clauses for the inner **IF** ranges of a compound **IF** statement in order to allow alternative branches for the outer **IF** statements. In this case it is not sufficient to follow the **ELSE** with a semicolon or a null clause. Instead, the **NOP** (no-operation) instruction can be used for this purpose (see **NOP** on page 10-61).

For example:

```
if i > 2 then say 'really?'  
    else say 'I thought so'
```

END

Usage: END [variable]

The **END** instruction terminates the range of a **DO** or **SELECT** instruction. If the optional variable symbol is supplied, it is compared to the index variable of the **DO** statement (which must therefore be iterative). An error is generated if the symbols do not match, so this provides a simple mechanism for matching the **DO** and **END** statements.

For example:

```
do i = 1 to 5                                /* index variable is i */  
    say i  
end i                                         /* end "i" loop */
```

EXIT

Usage: EXIT [expression]

The **EXIT** instruction terminates the execution of a program and is valid anywhere within a program. The evaluated expression is passed back to the caller as the function or command result.

Results Processing. The processing of the **EXIT** result depends on whether a result string was requested by the calling program and whether the current invocation resulted from a command or function call. If a result string was requested, the expression result is copied to a block of allocated memory and a pointer to the block is returned as the secondary result of the call.

If the caller did not request a result string and the program was invoked as a command, then an attempt is made to convert the expression result to an integer. This value is then returned as the primary result, with 0 as the secondary result. This allows the **EXIT** expression to be interpreted as a return code by the caller. Refer to Chapter 7, "Using AmigaDOS" for information on return codes.

For example:

```
exit                                /* no result needed */
exit 12                             /* an error return? */
```

IF

Usage: IF expression [THEN] [;] [conditional statement]

The **IF** instruction is used in conjunction with **THEN** and **ELSE** instructions to conditionally execute a statement. The result of the expression must be a boolean value. If the result is 1 (**True**), the statement following the **THEN** symbol is executed; otherwise, control passes to the next statement (which might be an **ELSE** clause). The **THEN** keyword need not immediately follow the **IF** expression, but may appear as a separate clause. The instruction is actually analyzed as "**If expression; THEN; statement;**". The expression following the **IF** statement establishes the test condition that determines whether subsequent **THEN** or **ELSE** clauses will be performed.

Any valid statement may follow the **THEN** symbol. In particular, a "**DO; . . . END;**" group allows a series of statements to be performed conditionally.

For example:

```
if result < 0 then exit          /* all done? */
```

INTERPRET

Usage: INTERPRET expression

The **INTERPRET** command treats the expression as though it were a source statement block in your program. The expression is evaluated and the result is executed as one or more program statements. The statements are considered as a group, as though surrounded by a "**DO; ... ; END**" combination. Any statements can be included in the **INTERPRET**ed source, including **DO** or **SELECT** instructions.

An **INTERPRET** instruction activates a control range when it is executed, which serves as a boundary for **LEAVE** and **ITERATE** instructions. These instructions can therefore be used only with **DO**-loops defined within the **INTERPRET**. The **BREAK** instruction can be used to terminate the processing of **INTERPRET**ed statements. While it is not an error to include label clauses within the interpreted string, only those labels defined in the original source code are searched during a transfer of control.

The **INTERPRET** instruction can be used to construct programs dynamically and then execute them. Program fragments may be passed as arguments to functions, which then **INTERPRET**s the fragments.

For example:

```
inst = 'say'          /* an instruction */  
interpret inst hello  /* ... "say HELLO" */
```

ITERATE

Usage: ITERATE [variable]

The **ITERATE** instruction terminates the current iteration of a **DO** instruction and begins the next iteration. Effectively, control passes to the **END** statement and then (depending on the outcome of the **UNTIL** expression) back to the **DO** statement. The instruction normally acts on the innermost iterative **DO** range containing the instruction. An error results if the **ITERATE** instruction is not contained within an iterative **DO** instruction.

The optional variable symbol specifies which **DO** range is to be exited, in the event that several nested ranges exist. The variable is taken as a literal and must match the index variable of a currently active **DO** instruction. An error results if no such matching **DO** instruction is found.

For example:

```
do i = 1 to 5
  if i = 3 then iterate i
  say i
end
→ 1, 2, 4, 5
```

LEAVE

Usage: LEAVE [variable]

LEAVE forces an immediate exit from the iterative **DO** range containing the instruction. An error results if the **LEAVE** instruction is not contained within an iterative **DO** instruction.

The optional variable symbol specifies which **DO** range is to be exited, in the event that several nested ranges exist. The variable is taken as a literal and must match the index variable of a currently active **DO** instruction. An error results if no such matching **DO** instruction is found.

For example:

```
do i = 1 to limit
  if i > 5 then leave      /* maximum iterations */
end
```

NOP

Usage: **NOP**

The **NOP** or “no-operation” instruction does just that: nothing. It is provided to control the binding of **ELSE** clauses in compound **IF** statements.

For example:

```
if i = j then              /* first (outer) IF */
  if j = k then a = 0      /* inner IF */
    else nop              /* binds to inner IF */
  else a = a + 1          /* binds to outer IF */
```

NUMERIC

Usage: **NUMERIC {DIGITS | FUZZ} *expression***
or: **NUMERIC FORM {SCIENTIFIC | ENGINEERING}**

This instruction sets options relating to the numeric precision and format. The valid forms of the **NUMERIC** instruction are:

- **NUMERIC DIGITS *expression***. Specifies the number of digits of precision for arithmetic calculations. The *expression* must evaluate to a positive whole number.
- **NUMERIC FUZZ *expression***. Specifies the number of digits to be ignored in numeric comparison operations. This must be a positive whole number that is less than the current **DIGITS** setting.
- **NUMERIC FORM SCIENTIFIC**. Specifies that numbers that require exponential notation be expressed in **SCIENTIFIC** notation. The exponent is adjusted so that the mantissa (for non-zero numbers) is between 1 and 10. This is the default format.

- **NUMERIC FORM ENGINEERING.** Selects **ENGINEERING** format for numbers that require exponential notation. **ENGINEERING** format normalizes a number so that its exponent is a multiple of three and the mantissa (if not 0) is between 1 and 1000.

The numeric options are preserved when an internal function is called.

For example:

```
numeric digits 12                /* precision */  
numeric form scientific          /* format */
```

OPTIONS

Usage: **OPTIONS** [*FAILAT expression*]
or: **OPTIONS** [*PROMPT expression*]
or: **OPTIONS** [*RESULTS*]

The **OPTIONS** instruction is used to set various internal defaults. The **FAILAT** expression sets the limit at or above which command return codes will be signalled as errors and it must evaluate to an integer value. The **PROMPT** expression provides a string to be used as the prompt with the **PULL** (or **PARSE PULL**) instruction. The **RESULTS** keyword indicates that the interpreter should request a result string when it issues commands to an external host.

The internal options controlled by this instruction are preserved across function calls, so an **OPTIONS** instruction can be issued within an internal function without affecting the caller's environment. If no keyword is specified with the **OPTIONS** instruction, all controlled options revert to their default settings.

For example:

```
options failat 10
options prompt "Yes Boss?"
options results
```

NOTE: The **OPTIONS** instruction now accepts a **NO** keyword to reset a selected option to its default value, making it more convenient to reset the **RESULTS** attribute for a single command without having to reset the **FAILAT** and **PROMPT** options. **OPTIONS** also accepts a new keyword **CACHE** that can be used to enable or disable an internal statement-caching scheme. The cache is normally enabled.

OTHERWISE

Usage: OTHERWISE [;] [conditional statement]

This instruction is valid only within the range of a **SELECT** instruction and must follow all of the "**WHEN . . . THEN**" statements. If none of the preceding **WHEN** clauses have succeeded, the statement following the **OTHERWISE** instruction is executed. An **OTHERWISE** is not mandatory within a **SELECT** range. However, an error will result if the **OTHERWISE** clause is omitted and none of the **WHEN** instructions succeed.

For example:

```
select
  when i = 1 then say 'one'
  when i = 2 then say 'two'
  otherwise say 'other'
end
```

PARSE

Usage: **PARSE** [**UPPER**] *inputsource* [*template*] [,*template* . . .]

The **PARSE** instruction provides a mechanism to extract one or more substrings from a string and assign them to variables. The input string can come from a variety of sources, including argument strings, an expression or from the console. The *template* provides both the variables to be given values and the way to determine the value strings. The template may be omitted if the instruction is intended only to create the input string. The different options of the instruction are described below.

Input Sources. The sources for the input strings are specified by the keyword symbols listed below. When multiple templates are supplied, each template receives a new input string, although for some source options the new string will be identical to the previous one. The input source string is copied before being parsed, so the original strings are never altered by the parsing process. For further information, turn to page 10-146.

- **UPPER.** This optional keyword may be used with any of the input sources and specifies that the input string is to be translated to uppercase before being parsed. It must be the first token following **PARSE**.
- **ARG.** This input option retrieves the argument strings supplied when the program was invoked. Command invocations normally have only a single argument string, but functions may have up to 15 argument strings.
- **EXTERNAL.** The input string is read from STDERR stream, (see page 10-138) so as not to disturb any PUSHed (see page 10-166) or QUEUED (see page 10-167)

data. If multiple templates are supplied, each template will read a new string. This source option is the same as **PULL**.

- **NUMERIC**. The current numeric options are placed in a string in the order **DIGITS**, **FUZZ** and **FORM**, separated by a single space.
- **PULL**. Reads a string from the input console. If multiple templates are supplied, each template will read a new string.
- **SOURCE**. The "source" string for the program is retrieved. This string is formatted as "{**COMMAND** | **FUNCTION**} {0 | 1} *called resolved ext host*." The first token indicates whether the program was invoked as a command or as a function. The second token is a boolean flag indicating whether a result string was requested by the caller. The *called* token is the name used to invoke this program, while the *resolved* token is the final resolved name of the program. The *ext* token is the file extension to be used for searching (the default is "REXX"). Finally, the host token is the initial host address for commands.
- **VALUE *expression* WITH**. The input string is the result of the supplied expression. The **WITH** keyword is required to separate the expression from the template. The expression result may be parsed repeatedly by using multiple templates, but the expression is not reevaluated.
- **VAR *variable***. The value of the specified variable is used as the input string. When multiple templates are provided, each template uses the current value of the variable. This value may change if the variable is included as an assignment target in any of the templates.

- **VERSION.** The current configuration of the AREXX interpreter is supplied in the form "**AREXX** *version cpu mpu video freq.*" The *version* token is the release level of the interpreter, formatted as **1.14**. The *cpu* token indicates the processor currently running the program, and will be one of the values 68000, 68010 or 68020. The *mpu* token will be either **NONE** or **68881** or **68882** depending on whether a math coprocessor is available on the system. The *video* token will indicate either **NTSC** or **PAL** and the *freq* token gives the clock (line) frequency as either **60Hz** or **50Hz**.

Parsing by Template

Parsing is controlled by a template, which may consist of symbols, strings, operators and parentheses. During the parsing operation the input string is split into substrings that are assigned to the variable symbols in the template. The process continues until all of the variables in the template have been assigned a value. If the input string is "used up", any remaining variables are given null values.

Templates are described in depth in a later section, so only a simplified description is presented here. The goal of the parsing operation is to associate a current and next position with each variable symbol in the template. The substring between these positions is then assigned as the value to the variable. There are three basic methods used to determine the value strings.

Parsing by Tokenization. When a variable in the template is followed immediately by another variable, the value string is determined by breaking the input string into words separated by blanks. Each word is assigned to a variable in the template.

Values determined by tokenization will never have leading or trailing blanks. Normally the last variable in the template receives the untokenized remainder of the input string, since it is not followed by a symbol. A placeholder symbol, a period (.), forces the variable with the period to terminate at the first space in the input stream. Placeholders behave like variables in the template except that they are never actually assigned a value.

For example:

```
/* Numeric string is: "9 0 SCIENTIFIC"          */
parse numeric digits fuzz form .
say digits                                     /* → 9 */
say fuzz                                       /* → 0 */
say form                                       /* → SCIENTIFIC */
```

Parsing by Position. If the fields in the input string have known positions, value strings can be specified by absolute or relative positions. Relative positions are indicated by a number preceded by a "+" or "-" operator. Each positional marker updates the scan position in the string. The value assigned to a variable is the string from the current position up to, but not including, the next position in the string.

For example:

```
myvar = 1234567890
parse var myvar 1 a 3 b +2 c 1 d
say a
say b
say c
say d
```

This is the output:

```
12
34
567890
1234567890
```

NOTE: The PARSE SOURCE option now returns the full path name of the AREXX program file. Formerly just a relative name was given, which was not sufficient to locate the program's source file.

PROCEDURE

Usage: **PROCEDURE** [*EXPOSE variable [variable . . .]*]

The **PROCEDURE** instruction is used within an internal function to create a new symbol table. This protects the symbols defined in the caller's environment from being altered by the execution of the function. **PROCEDURE** is usually the first statement within the function, although it is valid anywhere within the function body. It is an error to execute two **PROCEDURE** statements within the same function.

For example:

```
fact: procedure                                /* a recursive function */
  arg i
  if i = 1
    then return 1
    else return i*fact(i-1)
```

Exposing Variables. The **EXPOSE** subkeyword provides a selective mechanism for accessing the caller's symbol table, and for passing global variables to a function. The variables following the **EXPOSE** keyword are taken to refer to symbols in the caller's table. Any subsequent changes made to these variables will be reflected in the caller's environment.

The variables in the **EXPOSE** list may include stems or compound symbols, in which case the ordering of the variables becomes significant. The **EXPOSE** list is processed from left to right and compound symbols are expanded based on the values in effect in the new generation. For example, suppose that the value of the symbol **J** in the previous generation is **123**, and that **J** is uninitialized in the new generation. Then **PROCEDURE EXPOSE J A.J** will expose **J** and **A.123**, whereas **PROCEDURE EXPOSE A.J J** will expose **A.J** and **J**. Exposing a stem has the effect of exposing all possible compound symbols derived from that stem. That is, **PROCEDURE EXPOSE A.** exposes **A.I**, **A.J**, **A.J.J**, **A.123**, etc.

PULL

Usage: **PULL** [*template*] [*template* . . .]

This is a shorthand form of the **PARSE UPPER PULL** instruction. It reads a string from the input console, translates it to uppercase and parses it using the template. Multiple strings can be read by supplying additional templates. The instruction will read from the console even if no template is given.

Templates are described briefly with the **PARSE** instruction.

For example:

```
pull first last .
```

```
/* read names */
```


RETURN

Usage: **RETURN** [*expression*]

RETURN is used to leave a function and return control to the point of the previous function invocation. The evaluated expression is returned as the function result. If an expression is not supplied, an error may result in the caller's environment. Functions called from within an expression must return a result string and will generate an error if no result is available. Functions invoked by the **CALL** instruction need not return a result.

A **RETURN** issued from the base environment of a program is not an error and is equivalent to an **EXIT** instruction. Refer to the **EXIT** instruction for a description of how result strings are passed back to an external caller.

For example:

```
return 6*7                                /* the answer */
```

SAY

Usage: **SAY** [*expression*]

The result of the evaluated expression is written to the output console, with a "newline" character appended. If the expression is omitted, a null string is sent to the console.

For example:

```
say 'The answer is ' value
```

SELECT

Usage: **SELECT**

This instruction begins a group of instructions containing one or more **WHEN** clauses and possibly a single **OTHERWISE** clause, each followed by a conditional statement.

Only one of the conditional statements within the **SELECT** group will be executed. Each **WHEN** statement is executed in succession until one succeeds; if none succeeds, the **OTHERWISE** statement is executed. The **SELECT** range must be terminated by an eventual **END** statement.

For example:

```
select
  when i = 1 then say 'one'
  when i = 2 then say 'two'
  otherwise say 'other'
end
```

SHELL

Usage: **SHELL** [*symbol* | *string* | [*value*] [*expression*]]

The **SHELL** instruction is a synonym for the **ADDRESS** instruction.

For example:

```
shell edit                               /* set host to 'EDIT' */
```

SIGNAL

Usage: **SIGNAL** {**ON** | **OFF**} *condition*

or: **SIGNAL** [*VALUE*] *expression*

There are two forms of the **SIGNAL** instruction. The first form illustrated controls the state of the internal interrupt flags. Interrupts allow a program to detect and retain control when certain errors occur. In this form **SIGNAL** must be followed by one of the keywords **ON** or **OFF** and one of the condition keywords listed below. The interrupt flag specified by the condition symbol is then set to the indicated state. The valid signal conditions are:

- **BREAK_C** A "control-C" break was detected.
- **BREAK_D** A "control-D" break was detected.

- **BREAK_E** A "control-E" break was detected.
- **BREAK_F** A "control-F" break was detected.
- **ERROR** A host command returned a non-zero code.
- **HALT** An external **HALT** request was detected.
- **IOERR** An error was detected by the I/O system.
- **NOVALUE** An uninitialized variable was used.
- **SYNTAX** A syntax or execution error was detected.

The condition keywords are interpreted as labels to which control will be transferred if the selected condition occurs. For example, if the **ERROR** interrupt is enabled and a command returns a non-zero code, the interpreter will transfer control to the label **ERROR:**. The condition label must of course be defined in the program; otherwise, an immediate **SYNTAX** error results and the program exits.

In **SIGNAL [value] expression**, the tokens following **SIGNAL** are evaluated as an expression. An immediate interrupt is generated that transfers control to the label specified by the expression result. The instruction thus acts as a "computed goto".

Interrupts. Whenever an interrupt occurs, all currently active control ranges (**IF**, **DO**, **SELECT**, **INTERPRET**, or interactive **TRACE**) are dismantled before the transfer of control. Thus, the transfer cannot be used to jump into the range of a **DO**-loop or other control structure. Only the control structures in the current environment are affected by a **SIGNAL** condition, so it is safe to **SIGNAL** from within an internal function without affecting the state of the caller's environment.

Special Variables. The special variable **SIGL** is set to the current line number whenever a transfer of control occurs. The program can inspect **SIGL** to determine which line was being executed before the transfer. If an **ERROR** or **SYNTAX** condition causes an interrupt, the special variable **RC** is set to the error code that triggered the interrupt. For the **ERROR** condition, this code is usually an error severity level. Refer to page 10-157 for further details on **ERROR** codes and severity. The **SYNTAX** condition will always indicate an AREXX error code.

For example:

```
signal on error          /* enable interrupt */
signal off syntax       /* disable SYNTAX */
signal start            /* goto START */
```

WHEN

Usage: WHEN expression [THEN ;] [conditional statement]

The **WHEN** instruction is similar to the **IF** instruction, but is valid only within a **SELECT** range. Each **WHEN** expression is evaluated in turn and must result in a boolean value. If the result is a 1, the conditional statement is executed and control passes to the **END** statement that terminates the **SELECT**. As in the case of the **IF** instruction, the **THEN** need not be part of the same clause.

For example:

```
select;
  when i<j then say 'less'
  when i=j then say 'equal'
  otherwise say 'greater'
end
```


Commands

The REXX language is unusual in that an entire syntactic class of program statements is reserved for *commands*, statements that have meaning not within the language itself but rather to an external program. When a command clause is found in a program, it is evaluated as an expression and then sent through the *command interface* to an explicit or implicit *host application*, an external program that has announced its ability to receive commands. The host application then processes the command and returns a result code that indicates whether the command was performed successfully. In this manner every host program becomes fully programmable and even a limited set of predefined operations can be customized by the end user.

This section discusses the AREXX command interface and examines some of the ways in which commands can be used to build programs for an external program. Such programs are often called “macros” or “macro programs” because they implement a complex (“macro”) action from a series of simpler “micro” commands.

Command Clauses

Syntactically, a command clause is just an expression that can't be classified as another type of clause. The actual structure of the command is dictated by the external host to which it is intended, but in most cases will follow the model of a name or letter followed by parameter data.

Command names can be given as either a symbol or a string. However, it is generally safer to use a string for the name, since it can't be assigned a value or be mistaken for an instruction keyword.

For example, the following might be commands for a text editor:

```
JUMP current + 10      /* advance to next */  
'insert' newstring    /* place new string */  
'TOP'                 /* back to the top  */
```

Since command clauses are expressions, they are fully evaluated before being sent to the host. Any part of the final command string can be computed within the program, so virtually any sort of command structure can be created.

The interpretation of the received commands depends entirely on the host application. In the simplest case the command strings will correspond exactly to commands that could be entered directly by a user. For instance, positional control (up/down) commands for a text editor would probably have identical interpretations. Other commands may be valid only when issued from a macro program. A command to simulate a menu operation would probably not be entered by the user.

The Host Address

The destination for a command is determined by the current *host address*, which is the name of the *public message port* managed by an external program. AREXX maintains two implicit host addresses, a "current" and a "previous" value, as part of the program's storage environment. These values can be changed at any time using the **ADDRESS** instruction (or its synonym, **SHELL**) and the current host address can be inspected with the **ADDRESS()** Built-In function. The default host address string is "REXX", but this can be overridden when a program is invoked. In particular, most host applications will supply the name of their public port when they invoke a macro program, so that the macro can automatically issue commands back to the host.

One special host address is recognized. The string **COMMAND** indicates that the command should be issued directly to the underlying DOS. All other host addresses are assumed to refer to a public message port. An attempt to send a command to a nonexistent message port will generate the syntax error "Host environment not found".

Single commands can be sent to a specific host without disturbing the host address settings. This is done using the **ADDRESS** instruction, as the following example illustrates:

ADDRESS MYEDIT 'jump top'

This example would send the command "jump top" to an external host named "MYEDIT".

It is important to note that *you cannot send commands to a host application without knowing the name of its public message port*. Writing macro programs to communicate with two or more hosts may require some clever programming to determine whether both hosts are active and what their respective addresses are.

```
/* ED-status.rexx. Prints status of Ed. Ed must be running before this
   program is started. ED ports are named 'Ed', 'Ed_1', 'Ed_2', and so
   on. */
```

```
DEFAULT_ED = "Ed"
```

```
/* Procedure to follow if Ed isn't running, or if only a 2nd (or subsequent)
   instance of Ed is running.
   */
```

```
do while ~ show('p', DEFAULT_ED) /* Look for Ed's port */
    say "Cannot find port named" DEFAULT_ED
    say "Available ports:"
    say show('p') '0a'X
    say "Enter different name for port, or QUIT to quit"
    /* let user choose port if we can't find it */
    DEFAULT_ED = readln(stdout)
```

```
if strip(upper(DEFAULT_ED)) = 'QUIT' then exit 10 /* let user bail
                                                    out */

end
say "Using Ed port" DEFAULT_ED /* Message if successful */

/* Now that we've found the port, have AREXX address it. */
address value DEFAULT_ED

/* Set up some useful stem variables */
STEM.0 = 15 /* number of ED AREXX variables */
STEM.1 = 'LEFT' /* left margin (SL) */
STEM.2 = 'RIGHT' /* right margin (SR) */
STEM.3 = 'TABSTOP' /* tab stop setting (ST) */
STEM.4 = 'LMAX' /* max visible line on screen */
STEM.5 = 'WIDTH' /* width of scr in chars */
STEM.6 = 'X' /* physical X position on screen starting
              from 1 */
STEM.7 = 'Y' /* physical Y position on screen starting
              from 1 */
STEM.8 = 'BASE' /* Window base (0 unless screen shifted
                 right) */
STEM.9 = 'EXTEND' /* extended margin value (EX) */
STEM.10 = 'FORCECASE' /* case sensitivity flag */
STEM.11 = 'LINE' /* current line number, first line is 1 */
STEM.12 = 'FILENAME' /* file being edited */
STEM.13 = 'CURRENT' /* text of current line */
STEM.14 = 'LASTCMD' /* last extended command */
STEM.15 = 'SEARCH' /* last string searched for */

'rv' '/STEM/' /* ask ED to put values into stem variable
               'STEM.' */
```



```
/* Note that STEM.1 is LEFT, and STEM.LEFT now holds a value from Ed.  
Here is a way to print out that information.  
*/  
  
do i = 1 to STEM.0  
    ED_VAR = STEM.i  
    say STEM.i " = " STEM.ED_VAR /* print out Ed variable and value */  
end
```

The Command Interface

AREXX implements its command interface using the message-passing facilities provided by the Amiga's operating system. Each host application must provide a public message port, the name of which is referred to as the *host address*. AREXX programs issue commands by placing the command string in a message packet and sending the packet to the host's message port. The program suspends operation while the host processes the commands and resumes when the message packet returns. The entire process can be regarded as a dialogue between the host application and a macro (the AREXX program); the host initiates the dialogue by invoking the macro and the macro program replies with one or more command strings. The commands that can be sent are not limited to simple text strings, but might be address pointers or even bit-mapped images.

After it finishes processing a command, the host "replies" the message packet with a *return code* that indicates the status of the command. This return code is placed in the AREXX special variable **RC** so that it can be examined by the program. A value of zero is assumed to mean that no errors occurred, while positive values usually indicate progressively more severe error conditions. The return code allows the macro program to

determine whether the command succeeded and to take action if it failed, so it is important for each applications program to document the meanings of the return codes for its commands.

Using Commands in Macro Programs

AREXX can be used to write programs for any host application that includes a suitable command interface. Some applications programs are designed with an embedded macro language and may include many predefined macro commands.

The starting point in designing a macro program is to examine the commands that would be required to perform it manually. The documentation for the host application program should then describe the possible return codes for each command. These codes can be used to determine whether the operation performed by the command was successful. Check also for "shortcut" commands that may be available only to macro programs. Some applications programs may include very powerful functions that were implemented specifically for use in macro programs.

An example of an AREXX program called through the ED editor:

```
/* transpose.ed: transpose character */
/* Given string '123', if cursor is on 3, this macro converts */
/* string into '213'. */
HOST = address() /* find out which ED called us */
address value HOST /* . . . and talk to it */
'rv' '/CURR/' /* Have ED put info in stem CURR */
/* We'll need two pieces of */
/* information: */
```

```
currpos = CURR.X          /* position of cursor on line */
currlin = CURR.CURRENT    /* contents of current line */

if (currpos > 2) then      /* Must work on current line */
    currpos = currpos - 1
else do                   /* Report error and exit */
    'sm /Cursor must be at position 2 or further to the right/'
    exit 10
end

/* Need to reverse the CURRPOSth and CURRPOSth-1 characters and
   replace the current line with the new one. */

drop CURR.               /* stem variable CURR is no longer needed; save
                           some memory */

'd'                       /* Tell ED to delete current line */
currlin = swapch (currpos, currlin) /* swap the two characters */
'i' || currlin || '/'     /* insert modified line */
do i = 1 to currpos       /* place cursor back where it
                           started */
    'cr'                  /* ED's 'cursor right' command */
end

exit                      /* All done */

/* Function to swap two characters */
swapch: procedure
parse arg cpos,clin
    chl = substr (clin, cpos, 1) /* get character */
    clin = delstr (clin, cpos, 1) /* delete it from string */
    clin = insert (chl,clin,cpos-2,1) /* insert to create transposition */
return clin                /* return to modified string */
```

Using AREXX with Command Shells

Although AREXX was designed to work most effectively with programs that support its specific command interface, it can be used with any "command shell" program that uses standard I/O mechanisms to obtain its input stream. There are several ways to use AREXX to prepare a stream of commands for such a program.

You could create an actual command file on the RAM disk and then pass it directly to the Shell. For example, you could open a new Shell to run a standard "execute" script using the following short program:

```
/* Launch a new Shell */  
address command  
conwindow = "CON:0/0/640/100/NewOne/close"  
  
/* create a command file on the fly */  
call open out,"ram:temp",write  
call writeln out,'echo "this is a test"'  
call close out  
  
/* open the new Shell window */  
'newshell' conwindow "ram:temp"  
exit
```

Since no disk accesses are required, this method is fairly fast, if not very elegant.

Command Inhibition

Sometimes it is necessary to write and test macro programs that issue potentially destructive commands. For instance, a program to find and delete unneeded files would be difficult to test safely, since it might accidentally delete the wrong files and would require a continual source of new files for testing.

To simplify the development and testing of such programs, AREXX provides a special tracing mode called *command inhibition* that suppresses host commands. While in command inhibition mode, command processing proceeds normally except that the command is not actually issued and the variable **RC** is set to 0. This allows the program logic to be verified before any commands are actually sent to the external program.

See page 10-139 for information enabling the command inhibition.

Functions

The basic concept of a function is a program or group of statements that will be executed whenever the function name appears in a certain context. Functions are an important building block of most computer languages in that they allow *modular programming* — the ability to build large programs from a series of smaller, more easily developed modules. In AREXX a function may be defined as part of (internal to) a program, as part of a library or as a separate external program.

The Library List

The resident process maintains a Library List of the *function libraries* and *function hosts* currently available to AREXX programs. This list is used to resolve all references to external functions. Each entry has an associated search priority in the range 100 (highest) to -100 (lowest), with the higher-valued entries being searched first until the requested function is found. The list is searched by calling each entry, using the appropriate protocol, until the return code indicates that the function was found.

The two types of entities maintained by the list are quite different in some respects, but the ultimate way in which a function call is resolved is transparent to the calling program. A function library is a collection of functions organized as an Amiga shared library, while a function host is a separate task that manages a message port. Function libraries are called as part of the AREXX interpreter's task context, but calls to function hosts are mediated by passing a message packet. The AREXX resident process is itself a function host and is installed in the Library List at a priority of -60 .

The resident process provides addition and deletion operations for maintaining the Library List. These operations are performed by sending an appropriate message packet. The Library List is always maintained in priority order. Within a given priority level any new entries are added to the end of the chain, so that entries added first will be searched first. The priority levels are significant if any of the libraries have duplicate function name definitions, since the function located further down the search chain could never be called.

Function Libraries. Each function library entry in the Library List contains a library name, a search priority, an entry point offset and a version number. The library name must refer to a standard Amiga shared library residing in the system **LIBS**:

directory so that it can be loaded when needed. Function libraries can be created and maintained by users or applications developers.

The "query" function is the library entry point that is actually called by the interpreter. It must be specified as an integer offset (e.g. "-30") from the library base. The return code from the query call then indicates whether the desired function was found. If it was, the function is called with the parameters passed by the interpreter and the function result is returned to the caller. Otherwise, the search continues with the next entry in the list. In either case, even the library is closed to await the next call.

NOTE: Not every Amiga shared library can be used as a function library. Function libraries must have a special entry point to perform the dynamic linking required to access the functions from within AREXX. Each library should include documentation providing its version number and the integer offset to its "query" entry point.

Syntax and Search Order

Function calls in an expression are defined syntactically as a symbol or string followed immediately by an open parenthesis. The symbol or string (taken as a literal) specifies the *function name* and the open parenthesis begin the *argument list*. Between the opening and eventual closing parentheses are zero or more argument expressions, separated by commas, that supply the data being passed to the function.

For example:

```
CENTER ('title',20)
ADDRESS()
'AllocMem'(256*4,1)
```

are all valid function calls. Each argument expression is evaluated in turn and the resulting strings are passed as the argument list to the function. There is no limit to the number of arguments that may be passed to an internal function, but calls to Built-In or external functions are limited to a maximum of 15 arguments. Note that each argument expression, while often just a single literal value, can include arithmetic or string operations or even other function calls. Argument expressions are evaluated from left to right.

Functions can also be invoked using the **CALL** instruction. The syntax of this form is slightly different and is described on page 10-53. The **CALL** instruction can be used to invoke a function that may not return a value.

Search Order

Function linkages in AREXX are established at the time of the function call. A specific search order is followed until a function matching the name symbol or string is found. If the specified function cannot be located, an error is generated and the expression evaluation is terminated. The full search order is:

1. **Internal Functions.** The program source is examined for a label that matches the function name. If a match is found, a new storage environment is created and control is transferred to the label.
2. **Built-In Functions.** The Built-In function library is searched for the specified name. All of these functions are defined by uppercase names and the library has been specially organized to make the search as efficient as possible.

3. **Function Libraries and Function Hosts.** The available function libraries and function hosts are maintained in a prioritized list, which is searched starting at the highest priority until the requested function is found or the end of the list is reached. Each function library is opened and called at a special entry point to determine whether it contains a function matching the given name. Function hosts are called using a message-passing protocol similar to that used for commands and may be used as gateways for remote procedure calls to other machines in a network.
4. **External AREXX Programs.** The final search step is to check for an external AREXX program file by sending an invocation message to the AREXX resident process. The search always begins in the current directory and follows the same search path as the original AREXX program invocation. The name matching process is not case-sensitive.

Note that the function name-matching procedure may be case sensitive for some of the search steps but not for others. The matching procedure used in a function library or function host is left to the discretion of the applications designer. Functions defined with mixed-case names must be called using a string token, since symbol names are always translated to uppercase.

The full search order is followed whenever the function name is defined by a symbol token. However, the search for internal functions is bypassed if the name is specified by a string token. This allows internal functions to usurp the names of external functions, as in the following example:

```
CENTER:                                /* internal "CENTER" */  
arg string,length                      /* get arguments */  
length = min(length,60)               /* compute length */  
return 'CENTER'(string, length)
```

Here the Built-In function **CENTER()** has been replaced by an internal function after modifying the length argument.

Internal Functions. The interpreter creates a new storage environment when an internal function is called, so that the previous (caller's) environment is preserved. The new environment inherits the values from its predecessor, but subsequent changes to the environment variables do not affect the previous environment. The specific values preserved are:

- The current and previous host addresses.
- The **NUMERIC DIGITS**, **FUZZ** and **FORM** settings.
- The trace option, inhibit flag and interactive flag.
- The state of the interrupt flags defined by the **SIGNAL** instruction.
- The current prompt string as set by the **OPTIONS PROMPT** instruction.

The new environment does not automatically get a new symbol table, so initially all of the variables in the previous environment are available to the called function. The **PROCEDURE** instruction can be used to create a table and thereby protect the caller's symbol values. **PROCEDURE** may be used to allow the same variable name to be used in two different areas with two different values.

Execution of the internal function proceeds until a **RETURN** instruction is executed. At this point the new environment is dismantled and control resumes at the point of the function call. The expression supplied with the **RETURN** instruction is evaluated and passed back to the caller as the function result.

Built-In Functions. AREXX provides a substantial library of predefined functions as part of the language system. These functions are always available and have been optimized to

work with the internal data structures. In general the Built-In functions execute much faster than an equivalent interpreted function, so their usage is strongly recommended.

External Function Libraries. External function libraries provide a mechanism with which users and applications developers can extend the functionality of AREXX. A function library is a collection of one or more functions together with a "query" entry point that serves to match a name string with the appropriate function. External function libraries are supported as standard Amiga shared libraries and may be either memory or disk-resident. Disk resident libraries are loaded and opened as needed.

The AREXX resident process maintains a list, called the *Library List*, of the currently available function libraries and function hosts. Application programs can add or remove function libraries as required. The Library List is maintained as a priority-sorted queue and entries can be added at an appropriate priority to control the function name resolution. Libraries with higher priorities are searched first. Within a given priority level, those libraries added first are searched first.

During the search process the AREXX interpreter opens each library and calls its "query" entry point. The query function must then check to see whether the requested function name is in the library. If not, it returns a "function not found" error code and the search continues with the next library in the list. Function libraries are always closed after being checked so that the operating system can reclaim the memory space if required. Once the requested function has been found, it is called with the arguments passed by the interpreter and must return an error code and a result string.

Function Hosts. The name associated with a function host is the name of its public message port. Function calls are passed to the host as a message packet; it is then up to the individual

host to determine whether the specified function name is one that it recognizes. The name resolution is completely internal to the host, so function hosts provide a natural gateway mechanism for implementing remote procedure calls to other machines in a network.

The Clip List

The Clip List maintains a set of (name,value) pairs that may be used for a variety of purposes. Each entry in the list consists of a name and a value string and may be located by name. Since the Clip List is publicly accessible, it may be used as a general clipboard-like mechanism for intertask communication. In general, the names used should be chosen to be unique to an application to prevent unintended duplications with other programs. Any number of entries may be posted to the list.

One potential application for the Clip List is as a mechanism for loading predefined constants into an AREXX program. The language definition does not include a facility comparable to the "header file" preprocessor in the "C" language. However, consider a string in the Clip List of the form:

pi=3.14159; e=2.718; sqrt2=1.414 . . .

(i.e., a series of assignments separated by semicolons). In use, such a string could be retrieved by name using the built-in function **GETCLIP()** and then **INTERPRET**ed within the program. The assignment statements within the string would then create the required constant definitions. The following program fragment illustrates the process:

```
/* assume a string called "numbers" is available */
numbers = getclip('numbers')           /* case-sensitive */
interpret numbers                       /* . . . assignments */
```


More generally, the strings would not be restricted to contain only assignment statements, but could include any valid AREXX statements. The Clip List could thus provide a series of programs for initializations or other processing tasks.

The resident process supports addition and deletion operations for maintaining the Clip List. The names in the (name,value) pairs are assumed to be in mixed case and are maintained to be unique in the list. An attempt to add a string with an existing name will simply update the value string. The name and value strings are copied when an entry is posted to the list, so the program that adds an entry is not required to maintain the strings.

Entries posted to the Clip List remain available until explicitly removed. The Clip List is automatically released when the resident process exits.

The Built-In Function Library

Reference

Many of the Built-In functions have optional as well as required arguments. The optional arguments are shown in brackets and generally have a default value that is used if the argument is omitted.

Maximum Arguments. While internal functions can be called with a number of arguments, the Built-In functions (and external functions as well) are limited to a maximum of 15 arguments.

Pad and Optional Characters. For functions that accept a "pad" character argument (characters inserted to fill or create spaces), only the first character of the argument string is significant. If

a null string is supplied, the default padding character (usually a blank) will be used. Similarly, where an option keyword is specified as an argument, only the first character is significant. Option keywords may be given in uppercase or lowercase.

I/O Support Functions. AREXX provides functions for creating and manipulating external DOS files. The functions available at the present time are **OPEN()**, **CLOSE()**, **READCH()**, **READLN()**, **WRITECH()**, **Writeln()**, **EOF()**, **SEEK()** and **EXISTS()**. Files are referenced by a "logical name", a case-sensitive name that is assigned to a file when it is first opened. There is no theoretical limit to the number of files that may be open simultaneously, although memory availability will impose a practical limit. All open files are closed automatically when the program exits.

Bit-Manipulation Functions. The functions **BITCHG()**, **BITCLR()**, **BITCOMP()**, **BITSET()** and **BITTST()** are provided to implement extended bit-testing on character strings. These functions differ from similar string-manipulation functions in that the elementary unit of comparison is the bit rather than the byte. Bit numbers are defined such that bit 0 is the low-order bit of the rightmost byte of the string.

Built-in Functions

ABBREV()

Usage: **ABBREV** (*string1*,*string2*,*[length]*)

Returns a boolean value that indicates whether *string2* is an abbreviation of *string1* with length greater than or equal to the specified *length* argument. The default length is 0, so the null string is an acceptable abbreviation.

For example:

```
say abbrev('fullname', 'ful')    → 1
say abbrev('almost', 'alm', 4)   → 0
say abbrev('any', '')            → 1
```

ABS()

Usage: **ABS**(*number*)

Returns the absolute value of the *number* argument which must be numeric.

For example:

```
say abs(-5.35)    → 5.35
say abs(10)       → 10
```

ADDLIB()

Usage: **ADDLIB**(*name*, *priority*, [*offset*, *version*])

Adds a function library or a function host to the Library List maintained by the resident process. The *name* argument specifies either the name of a function library or the public message port associated with a function host. The name is case-sensitive and any libraries thus declared should reside in the system **LIBS:** directory. The *priority* argument specifies the search priority and must be an integer between 100 and -100, inclusive. The *offset* and *version* arguments apply only to libraries. The *offset* is the integer offset to the library's "query" entry point and the *version* is an integer specifying the minimum acceptable release level of the library.

The function returns a boolean result that indicates whether the operation was successful. Note that if a library is specified, it is not actually opened at this time. Similarly no check is performed as to whether a specified function host port has been opened yet.

For example:

```
say addlib('rexsupport.library',0,-30,0) → 1
call addlib "EtherNet", -20             /* a gateway */
```

ADDRESS()

Usage: ADDRESS()

Returns the current host address string. The host address is the message port to which commands will be sent. The **SHOW()** function can be used to check whether the required external host is actually available. See also **SHOW()**.

For example:

```
say address() → REXX
```

ARG()

Usage: ARG([*number*],[*'Exists'* | *'Omitted'*])

ARG() returns the number of arguments supplied to the current environment. If the *number* parameter alone is supplied, the corresponding argument string is returned. If a number and one of the keywords **Exists** or **Omitted** is given, the boolean return indicates the status of the corresponding argument. Note that the existence or omission test does not indicate whether the string has a null value, but only whether a string was supplied.

For example:

```
/* Assume arguments were: ('one'.,10) */
say arg() → 3
say arg(1) → one
say arg(2,'O') → 1
```


B2C()

Usage: **B2C(string)**

Converts a string of binary digits (0,1) into the corresponding (packed) character representation. The conversion is the same as though the argument string had been specified as a literal binary string (e.g. '1010'B). Blanks are permitted in the string, but only at byte boundaries. This function is particularly useful for creating strings that are to be used as bit masks. See also **X2C()**.

For example:

say b2c('00110011')	→ 3
say b2c('01100001')	→ a

BITAND()

Usage: **BITAND(string1,string2,[pad])**

The argument strings are logically ANDed together, with the length of the result being the longer of the two operand strings. If a pad character is supplied, the shorter string is padded on the right; otherwise, the operation terminates at the end of the shorter string and the remainder of the longer string is appended to the result.

For example:

bitand('0313'x,'FFF0'x)	→ '0310'x
-------------------------	-----------

BITCHG()

Usage: **BITCHG(string,bit)**

Changes the state of the specified bit in the argument string. Bit numbers are defined such that bit 0 is the low-order bit of the rightmost byte of the string.

For example:

bitchg('0313'x,4)	→ '0303'x
-------------------	-----------

BITCLR()

Usage: BITCLR(*string*,*bit*)

Clears (sets to zero) the specified bit in the argument string. Bit numbers are defined such that bit 0 is the low-order bit of the rightmost byte of the string.

For example:

```
bitclr('0313'x,4)          → '0303'x
```

BITCOMP()

Usage: BITCOMP(*string1*,*string2*,*[pad]*)

Compares the argument strings bit-by-bit, starting at bit number 0. The returned value is the bit number of the first bit in which the strings differ or -1 if the strings are identical.

For example:

```
bitcomp('7F'x,'FF'x)       → 7
/* seventh bit */
bitcomp('FF'x,'FF'x)       → -1
```

BITOR()

Usage: BITOR(*string1*,*string2*,*[pad]*)

The argument strings are logically ORed together, with the length of the result being the longer of the two operand strings. If a *pad* character is supplied, the shorter string is padded on the right; otherwise, the operation terminates at the end of the shorter string and the remainder of the longer string is appended to the result.

For example:

```
bitor('0313'x,'003F'x)     → '033F'x
```

BITSET()

Usage: **BITSET**(*string*,*bit*)

Sets the specified bit in the argument string to 1. Bit numbers are defined such that bit 0 is the low-order bit of the rightmost byte of the string.

For example:

`bitset('0313'x,2)` → '0317'x

BITTST()

Usage: **BITTST**(*string*,*bit*)

The boolean return indicates the state of the specified bit in the argument string. Bit numbers are defined such that bit 0 is the low-order bit of the rightmost byte of the string.

For example:

`bittst('0313'x,4)` → 1

BITXOR()

Usage: **BITXOR**(*string1*,*string2*,*[pad]*)

The argument strings are logically exclusively-ORed together, with the length of the result being the longer of the two operand strings. If a *pad* character is supplied, the shorter string is padded on the right; otherwise, the operation terminates at the end of the shorter string and the remainder of the longer string is appended to the result.

For example:

`bitxor('0313'x,'001F'x)` → '030C'x

C2B()

Usage: C2B(*string*)

Converts the character string into the equivalent string of binary digits. See also C2X().

For example:

say c2b('abc') → 011000010110001001100011

C2D()

Usage: C2D(*string*,[*n*])

Converts the *string* argument from its character representation to the corresponding decimal number, expressed as ASCII digits (0-9). If *n* is supplied, the character string is considered to be a number expressed in *n* bytes. The string is truncated or padded with nulls on the left as required and the sign bit is extended for the conversion.

For example:

say c2d('0020'x)	→ 32
say c2d('FFFF ffff'x)	→ -1
say c2d('FF0100'x,2)	→ 256

C2X()

Usage: C2X(*string*)

Converts the *string* argument from its character representation to the corresponding hexadecimal number, expressed as the ASCII characters 0-9 and A-F. See also C2B().

For example:

say c2x('abc') → 616263

CENTER() or CENTRE()

Usage: **CENTER**(*string*,*length*,*[pad]*) or
CENTRE(*string*,*length*,*[pad]*)

Centers the *string* argument in a string with the specified *length*. If the length is longer than that of the string, *pad* characters or blanks are added as necessary.

For example:

say center('abc',6)	→ ' abc '
say center('abc',6,'+')	→ '+abc++'
say center('123456',3)	→ '234'

CLOSE()

Usage: **CLOSE**(*file*)

Closes the file specified by the given logical name. The returned value is a boolean success flag and will be 1 unless the specified file was not open.

For example:

say close('input')	→ 1
--------------------	-----

COMPRESS()

Usage: **COMPRESS**(*string*,*[list]*)

If the *list* argument is omitted, the function removes leading, trailing or embedded blank characters from the *string* argument. If the optional *list* is supplied, it specifies the characters to be removed from the string.

For example:

say compress(' why not ')	→ whynot
say compress('++12-34-+', '+-')	→ 1234

COMPARE()

Usage: **COMPARE**(*string1*,*string2*,*[pad]*)

Compares two strings and returns the index of the first position in which they differ or 0 if the strings are identical. The shorter string is padded as required using the supplied character or blanks.

For example:

say compare('abcde', 'abcce')	→ 4
say compare('abcde', 'abcde')	→ 0
say compare('abc + +', 'abc + -', '+')	→ 5

COPIES()

Usage: **COPIES**(*string*,*number*)

Creates a new string by concatenating the specified number of copies of the original. The *number* argument may be zero, in which case the null string is returned.

For example:

say copies('abc', 3)	→ abcabcabc
----------------------	-------------

D2C()

Usage: **D2C**(*number*)

Creates a string whose value is the binary (packed) representation of the given decimal number.

For example:

d2c(65)	→ '41'x /* the letter 'A' */
---------	------------------------------

D2X()

Usage: D2X(*number*,*[digits]*)

Converts a decimal number to hexadecimal.

For example:

d2x(31) → 1F

DATE()

Usage: DATE(*[option]*,*[date]*,*[format]*)

Returns the current date in the specified format. The default ('Normal') option returns the date in the form DD MMM YYYY, as in 20 Apr 1988. The options recognized are:

Basedate	— the number of days since January 1, 0001
Century	— the number of days since January 1 of the century
Days	— the number of days since January 1 of the current year
European	— the date in the form DD/MM/YY
Internal	— internal system days
Julian	— the date in the form YYDDD
Month	— the current month (in mixed case)
Normal	— the date in the form DD MMM YYYY
Ordered	— the date in the form YY/MM/DD
Sorted	— the date in the form YYYYMMDD
USA	— the date in the form MM/DD/YY
Weekday	— the day of the week (in mixed case)

These options can be shortened to just the first character.

In addition to its primary option, the DATE() function also accepts optional second and third arguments to supply the date either in the form of internal system days or in the 'sorted' form YYYYMMDD. The second argument is an integer specifying either system days (the default) or a sorted date. The

third argument specifies the form of the date and can be either 'I' or 'S'. The current date in system days can be retrieved using DATE('Internal').

For example:

say date()	→ 20 Apr 1988
say date('M')	→ April
say date(s)	→ 19880420
say date('s',date('i') + 21)	→ 19880511
say date('w',19890609,'S')	→ Friday

DATATYPE()

Usage: DATATYPE(*string*,[*option*])

If the *option* parameter is not specified, DATATYPE() tests whether the *string* parameter is a valid number and returns either NUM or CHAR. If an option keyword is given, the boolean return indicates whether the string satisfied the requested test.

The following option keywords are recognized:

DATATYPE() Options	
Keyword	Characters Accepted
Alphanumeric	Alphabetics (A-Z,a-z) or Numeric (0-9)
Binary	Binary Digits String
Lowercase	Lowercase Alphabetics (a-z)
Mixed	Mixed Upper/Lowercase
Numeric	Valid Numbers
Symbol	Valid REXX Symbols
Upper	Uppercase Alphabetics (A-Z)
Whole	Integer Numbers
X	Hex Digits String

For example:

say datatype('123')	→ NUM
say datatype('1a f2', 'x')	→ 1
say datatype('aBcde', 'L')	→ 0

DELSTR()

Usage: DELSTR(*string*,*n*,*[length]*)

Deletes the substring of the *string* argument beginning with the *n*th character for the specified *length* in characters. The default length is the remaining length of the string.

For example:

say delstr('123456', 2, 3)	→ 156
----------------------------	-------

DELWORD()

Usage: DELWORD(*string*,*n*,*[length]*)

Deletes the substring of the *string* argument beginning with the *n*th word for the specified *length* in words. The default length is the remaining length of the string. The deleted string includes any trailing blanks following the last word.

For example:

say delword('Tell me a story', 2, 2)	→ 'Tell story'
say delword('one two three', 3)	→ 'one two '

DIGITS()

Usage: DIGITS()

Returns the current numeric digits setting.

For example:

numeric digits 6	
say digits()	→ 6

EOF()

Usage: EOF(*file*)

Checks the specified logical file name and returns the boolean value 1 (True), if the end-of-file has been reached and 0 (False) otherwise.

For example:

say eof(infile) → 1

ERRORTXT()

Usage: ERRORTXT(*n*)

Returns the error message associated with the specified AREXX error code. The null string is returned if the number is not a valid error code.

For example:

say errortxt(41) → Invalid expression

EXISTS()

Usage: EXISTS(*filename*)

Tests whether an external file of the given *filename* exists. The name string may include device and directory specifications.

For example:

say exists('sys:c/ed') → 1

EXPORT()

Usage: EXPORT(*address*, [*string*], [*length*], [*pad*])

Copies data from the optional string into a previously-allocated memory area, which must be specified as a 4-byte address. The *length* parameter specifies the maximum number of characters to be copied. The default is the length of the string. If the

specified length is longer than the string, the remaining area is filled with the *pad* character or nulls ('00'x). The returned value is the number of characters copied.

Use caution with this function! Any area of memory can be overwritten, possibly causing a system crash. Task switching is forbidden while the copy is being done, so system performance may be degraded if long strings are copied. See also **IMPORT()** and **STORAGE()**.

For example:

```
count = export('0004 0000'x, 'The answer')
```

FORM()

Usage: **FORM()**

Returns the current numeric form setting.

For example:

```
numeric form SCIENTIFIC  
say form()                → SCIENTIFIC
```

FIND()

Usage: **FIND(string,phrase)**

The **FIND()** function locates a phrase of words in a larger string of words and returns the word number of the matched position.

For example:

```
say find('Now is the time', 'is the') → 2
```

FREESPACE()

Usage: **FREESPACE(address,length)**

Returns a block of memory of a given length to the interpreter's internal pool. The *address* argument must be a 4 byte string obtained by a prior call to **GETSPACE()**, the internal

allocator. It is not always necessary to release internally allocated memory, since it will be released to the system when the program terminates. However, if a very large block has been allocated, returning it to the pool may avoid memory space problems. The return value is a boolean success flag. See also **GETSPACE()**.

For example:

```
say freespace('00042000'x,32) → 1
```

NOTE: Calling **FREESPACE()** with no arguments will return the amount of memory available in the interpreter's internal pool.

FUZZ()

Usage: **FUZZ()**

Returns the current numeric fuzz setting.

For example:

```
numeric fuzz 3  
say fuzz() → 3
```

GETCLIP()

Usage: **GETCLIP(name)**

Searches the Clip List for an entry matching the supplied *name* parameter and returns the associated value string. The name matching is case-sensitive and the null string is returned if the name cannot be found. See also **SETCLIP()**.

For example:

```
/* Assume 'numbers' contains 'PI=3.14159' */  
say getclip('numbers') → PI=3.14159
```


GETSPACE()

Usage: **GETSPACE**(*length*)

Allocates a block of memory of the specified length from the interpreter's internal pool. The returned value is the 4-byte address of the allocated block, which is not cleared or otherwise initialized. Internal memory is automatically returned to the system when the AREXX program terminates, so this function should not be used to allocate memory for use by external programs. The Support Library includes the function **ALLOCMEM**(), which allocates memory from the system free list. See also **FREESPACE**().

For example:

```
say c2x(getspace(32))           → '0003BF40'x
```

HASH()

Usage: **HASH**(*string*)

Returns the hash attribute of a string as a decimal number and updates the internal hash value of the string.

For example:

```
say hash('1')                  → 49
```

IMPORT()

Usage: **IMPORT**(*address*, [*length*])

Creates a string by copying data from the specified 4-byte address. If the length parameter is not supplied, the copy terminates when a null byte is found. See also **EXPORT**().

For example:

```
extval = import('0004 0000'x,8)
```

INDEX()

Usage: INDEX(*string*,*pattern*,[*start*])

Searches for the first occurrence of the *pattern* argument in the *string* argument, beginning at the specified *start* position. The default start position is 1. The returned value is the index of the matched pattern or 0 if the pattern was not found.

For example:

```
say index("123456","23")      → 2
say index("123456","77")      → 0
say index("123123","23",3)    → 5
```

INSERT()

Usage: INSERT(*new*,*old*,[*start*],[*length*],[*pad*])

Inserts the *new* string into the *old* string after the specified *start* position. The default starting position is 0. The new string is truncated or padded to the specified *length* as required, using the supplied pad character or blanks. If the start position is beyond the end of the old string, the old string is padded on the right.

For example:

```
say insert('ab','12345')      → ab12345
say insert('123','+++',3,5,'-') → ++-123--
```

LASTPOS()

Usage: LASTPOS(*pattern*,*string*,[*start*])

Searches backwards for the first occurrence of the *pattern* argument in the *string* argument, beginning at the specified *start* position. The default starting position is the end of the string. The returned value is the index of the matched pattern or 0 if the pattern was not found.

For example:

say lastpos('2','1234')	→ 2
say lastpos('2','1234234')	→ 5
say lastpos('2','123234',3)	→ 2
say lastpos('2','13579')	→ 0

LEFT()

Usage: **LEFT**(*string*,*length*,*[pad]*)

Returns the leftmost substring in the given *string* argument with the specified *length*. If the substring is shorter than the requested length, it is padded on the right with the supplied *pad* character or blanks.

For example:

say left('123456',3)	→ 123
say left('123456',8,'+')	→ 123456++

LENGTH()

Usage: **LENGTH**(*string*)

Returns the length of the string.

For example:

say length('three')	→ 5
---------------------	-----

LINES()

Usage: **LINES**(*file*)

Returns the number of lines queued or typed ahead at the logical file, which must refer to an interactive stream. The line count is obtained as the secondary result of a **WaitForChar()** call.

For example:

push 'a line'	
push 'another one'	
say lines(stdin)	→ 2

MAX()

Usage: **MAX**(*number,number[,number, ...]*)

Returns the maximum of the supplied arguments, all of which must be numeric. At least two parameters must be supplied.

For example:

say max(2.1,3,-1) → 3

MIN()

Usage: **MIN**(*number,number[,number, ...]*)

Returns the minimum of the supplied arguments, all of which must be numeric. At least two parameters must be supplied.

For example:

say min(2.1,3,-1) → -1

OPEN()

Usage: **OPEN**(*file,filename,['Append' | 'Read' | 'Write']*)

Opens an external file for the specified operation. The *file* argument defines the logical name by which the file will be referenced. The *filename* is the external name of the file and may include device and directory specifications. The function returns a boolean value that indicates whether the operation was successful. There is no limit to the number of files that can be open simultaneously and all open files are closed automatically when the program exits. See also **CLOSE()**, **READ()**, and **WRITE()**.

For example:

say open('MyCon','CON:160/50/320/100/MyCon/cds') → 1
say open('outfile','ram:temp','W') → 1

OVERLAY()

Usage: OVERLAY(*new,old,[start],[length],[pad]*)

Overlays the *new* string onto the *old* string beginning at the specified *start* position, which must be positive. The default starting position is 1. The new string is truncated or padded to the specified *length* as required, using the supplied *pad* character or blanks. If the start position is beyond the end of the old string, the old string is padded on the right.

For example:

```
say overlay('bb', 'abcd')           → bbcd
say overlay('4', '123',5,5,'-')     → 123-4----
```

POS()

Usage: POS(*pattern,string,[start]*)

Searches for the first occurrence of the *pattern* argument in the *string* argument, beginning at the position specified by the *start* argument. The default starting position is 1. The returned value is the index of the matched string or 0 if the pattern wasn't found.

For example:

```
say pos('23', '123234')             → 2
say pos('77', '123234')             → 0
say pos('23', '123234',3)           → 4
```

PRAGMA()

Usage: PRAGMA(*option,[value]*)

This function allows a program to change various attributes relating to the system environment within which the program executes. The *option* argument is a keyword that specifies an

environmental attribute. The currently implemented options are listed below. The *value* argument supplies the new attribute value to be installed. The value returned by the function depends on the attribute selected. Some attributes return the previous value installed, while others may simply set a boolean success flag. The currently defined option keywords are listed below.

- **Directory.** Specifies a new "current" directory. The current directory is used as the "root" for filenames that do not explicitly include a device specification. The return is the old directory name. `PRAGMA('D')` is equivalent to `PRAGMA('D','')`; it returns the pathname of the current directory without changing the directory.
- **Priority.** Specifies a new task priority. The priority value must be an integer in the range -128 to 127, but the practical range is much more limited. AREXX programs should never be run at a priority higher than that of the resident process, which currently runs at priority 4. The returned value is the previous priority level.
- **Id.** Returns the task ID (the address of the task block) as an 8-byte hex string. The task ID is a unique identifier of the particular AREXX invocation and may be used to create a unique name for it.
- **Stack.** An option issued with the **PRAGMA** function that specifies a new stack value for your current AREXX program. (**Stack**, also an AmigaDOS command, is defined on page 8-122 in Chapter 8, *"AmigaDOS Reference."*) Note that when a new **Stack** value is declared (8092) the previous **Stack** value is returned (4000).

`PRAGMA('W',{ 'Null' | 'WorkBench' })` controls the task's WindowPtr field. Setting it to 'Null' will suppress any requestors that might otherwise be generated by a DOS call.

PRAGMA('**',[name]) defines the specified logical name as the current (""") console handler, thereby allowing the user to open two streams on one window. If the name is omitted, the console handler is set to that of the client's process.

Examples:

say pragma('D','df0:c')	→ Extras:
say pragma('D','df1:c')	→ WorkBench:c
say pragma('priority',-5)	→ 0
say pragma('Id')	→ 00221ABC
call pragma '**',STDOUT	
say pragma ('Stack', 8092)	→ 4000

RANDOM()

Usage: **RANDOM**([min],[max],[seed])

Returns a pseudo-random integer in the interval specified by the *min* and *max* arguments. The default minimum value is 0 and the default maximum value is 999. The interval **max-min** must be less than or equal to 1000. If a greater range of random integers is required, the values from the **RANDU**() function can be suitably scaled and translated.

The *seed* argument can be supplied to initialize the internal state of the random number generator. See also **RANDU**().

For example:

```
thisroll = random(1,6)           /* might be 1 */
nextroll = random(1,6)           /* snake eyes? */
```

RANDU()

Usage: **RANDU**([seed])

Returns a uniformly-distributed, pseudo-random number between 0 and 1. The number of digits of precision in the result is always equal to the current Numeric Digits setting. With the choice of suitable scaling and translation values, **RANDU**() can be used to generate pseudorandom numbers on an arbitrary interval.

The optional integer *seed* argument is used to initialize the internal state of the random number generator. See also **RANDOM()**.

For example:

```
firsttry = randu()           /* 0.371902021? */  
numeric digits 3  
tryagain = randu()          /* 0.873? */
```

READCH()

Usage: **READCH**(*file,length*)

Reads the specified number of characters from the given logical file into a string. The length of the returned string is the actual number of characters read and may be less than the requested length if, for example, the end-of-file was reached. See also **READLN()**.

For example:

```
instring = readch('input',10)
```

READLN()

Usage: **READLN**(*file*)

Reads characters from the given logical file into a string until a "newline" character is found. The returned string does not include the "newline". See also **READCH()**.

For example:

```
instring = readln('MyFile')
```

REMLIB()

Usage: **REMLIB**(*name*)

Removes an entry with the given *name* from the Library List maintained by the resident process. The boolean return is 1 if the entry was found and successfully removed. Note that this

function does not make a distinction between function libraries and function hosts, but simply removes a named entry. See also **ADDLIB()**.

For example:

```
say remlib('MyLibrary.library')           → 1
```

REVERSE()

Usage: **REVERSE**(*string*)

Reverses the sequence of characters in the string.

For example:

```
say reverse('?ton yhw')                   → why not?
```

RIGHT()

Usage: **RIGHT**(*string,length,[pad]*)

Returns the rightmost substring in the given *string* argument with the specified *length*. If the substring is shorter than the requested length, it is padded on the left with the supplied *pad* character or blanks.

For example:

```
say right('123456',4)                     → 3456
say right('123456',8,'+')                 → + + 123456
```

SEEK()

Usage: **SEEK**(*file,offset,['Begin' | 'Current' | 'End']*)

Moves to a new position in the given logical file, specified as an offset from an anchor position. The default anchor is **Current**. The returned value is the new position relative to the start of the file.

For example:

```
say seek('input',10,'B')                   → 10
say seek('input',0,'E')                    → 356 /* file length */
```

SETCLIP()

Usage: SETCLIP(*name*, [*value*])

Adds a name-value pair to the Clip List maintained by the resident process. If an entry of the same name already exists, its value is updated to the supplied value string. Entries may be removed by specifying a null value. The function returns a boolean value that indicates whether the operation was successful.

For example:

```
say setclip('path', 'df0:s')    → 1
say setclip('path')             → 1
```

SHOW()

Usage: SHOW(*option*, [*name*], [*pad*])

Returns the names in the resource list specified by the *option* argument, or tests to see whether an entry with the specified *name* is available. The currently implemented options keywords are **Clip**, **Files**, **Libraries** and **Ports**, which are described below.

- **Clip**. Examines the names in the Clip List.
- **Files**. Examines the names of the currently open logical file names.
- **Libraries**. Examines the names in the Library List, which are either function libraries or function hosts.
- **Ports**. Examines the names in the system Ports List.

If the *name* argument is omitted, the function returns a string with the resource names separated by a blank space or the *pad* character, if one was supplied. If the *name* argument is given, the returned boolean value indicates whether the name was found in the resource list. The name entries are case-sensitive.

SIGN()

Usage: **SIGN**(*number*)

Returns 1 if the *number* argument is positive or zero and -1 if *number* is negative. The argument must be numeric.

For example:

say sign(12)	→ 1
say sign(-33)	→ -1

SOURCELINE()

Usage: **SOURCELINE**([*line*])

Returns the text for the specified line of the currently executing AREXX program. If the line argument is omitted, the function returns the total number of lines in the file. This function is often used to embed "help" information in a program.

For example:

/* A simple test program */	
say sourceline()	→ 3
say sourceline(1)	→ /* A simple test program */

SPACE()

Usage: **SPACE**(*string*,*n*,[*pad*])

Reformats the *string* argument so that there are *n* spaces (blank characters) between each pair of words. If the *pad* character is specified, it is used instead of blanks as the separator character. Specifying *n* as 0 will remove all blanks from the string.

For example:

say space('Now is the time',3)	→ 'Now is the time'
say space('Now is the time',0)	→ 'Nowisthetime'
say space('1 2 3',1,'+')	→ '1 + 2 + 3'

STORAGE()

Usage: **STORAGE**(*[address]*,*[string]*,*[length]*,*[pad]*)

Calling **STORAGE**() with no arguments returns the available system memory. If the *address* argument is given, it must be a 4-byte string and the function copies data from the (optional) *string* to the indicated memory address. The *length* parameter specifies the maximum number of bytes to be copied and defaults to the length of the string. If the specified length is longer than the string, the remaining area is filled with the *pad* character or nulls ('00'x).

The returned value is the previous contents of the memory area. This can be used in a subsequent call to restore the original contents.

Use caution with this function. Any area of memory can be overwritten, possibly causing a system crash. Task switching is forbidden while the copy is being done, so system performance may be degraded if long strings are copied. See also **EXPORT**()

For example:

```
say storage()           → 248400
oldval = storage('0004 0000'x,'The answer')
call storage '0004 0000'x,,32,' + '
```

STRIP()

Usage: **STRIP**(*string*,*[{'B' | 'L' | 'T'}]*,*[pad]*)

If neither of the optional parameters is supplied, the function removes both leading and trailing blanks from the *string* argument. The second argument specifies whether **Leading**, **Trailing** or **Both** (leading and trailing) characters are to be removed. The optional *pad* (or unpad, perhaps) argument selects the character to be removed.

For example:

```
say strip(' say what? ')      → 'say what?'
say strip(' say what? ', 'L')  → 'say what? '
say strip(' + + 123 + + + ', 'B', ' + ') → '123'
```

SUBSTR()

Usage: SUBSTR(*string*,*start*,*[length]*,*[pad]*)

Returns the substring of the *string* argument beginning at the specified *start* position for the specified *length*. The starting position must be positive and the default length is the remaining length of the string. If the substring is shorter than the requested length, it is padded on the right with the blanks or the specified *pad* character.

For example:

```
say substr('123456',4,2)      → 45
say substr('myname',3,6,' = ') → name = =
```

SUBWORD()

Usage: SUBWORD(*string*,*n*,*[length]*)

Returns the substring of the *string* argument beginning with the *n*th word for the specified *length* in words. The default length is the remaining length of the string. The returned string will never have leading or trailing blanks.

For example:

```
say subword('Now is the time ',2,2) → is the
```

SYMBOL()

Usage: SYMBOL(*name*)

Tests whether the *name* argument is a valid AREXX symbol. If the name is not a valid symbol, the function returns the string **BAD**. Otherwise, the returned string is **LIT** if the symbol is uninitialized and **VAR** if it has been assigned a value.

For example:

say symbol('J')	→ VAR
say symbol('x')	→ LIT
say symbol(' + + ')	→ BAD

TIME()

Usage: **TIME(option)**

Returns the current system time or controls the internal elapsed time counter. The valid option keywords are listed below.

TIME() Options	
Option Keyword	Description
Civil	Current time in 12 hour format (a.m./p.m.) hours/minutes
Elapsed	Elapsed time in seconds
Hours	Current time in hours since midnight
Minutes	Current time in minutes since midnight
Normal	Current time in 24 hour format hours/minutes/seconds
Reset	Reset the elapsed time clock
Seconds	Current time in seconds since midnight

If no option is specified, the function returns the current system time in the form HH:MM:SS.

For example:

```
/* Suppose that the time is 1:02 AM . . . */
say time('C')           → 5:46 PM
say time('Hours')        → 1
say time('m')            → 62
say time('n')            → 17:46:54
say time('S')            → 3720
call time 'R'            /* reset timer */
say time('E')            → .020
say time                 → 01:02:00
```

TRACE()

Usage: **TRACE**(*option*)

Sets the tracing mode to that specified by the *option* keyword, which must be one of the valid alphabetic or prefix options. The **TRACE()** function will alter the tracing mode even during interactive tracing, when **TRACE** instructions in the source program are ignored. The returned value is the mode in effect before the function call. This allows the previous trace mode to be restored later.

For example:

```
/* Assume tracing mode is ?ALL */
say trace('Results')    → ?A
```

TRANSLATE()

Usage: **TRANSLATE**(*string*, [*output*], [*input*], [*pad*])

This function constructs a translation table and uses it to replace selected characters in the argument string. If only the *string* argument is given, it is translated to uppercase. If an *input* table is supplied, it modifies the translation table so that characters in the argument string that occur in the input table are replaced with the corresponding character in the *output* table. Characters beyond the end of the output table are replaced with the specified *pad* character or a blank.

Note that the result string is always of the same length as the original string. The input and output tables may be of any length.

For example:

```
say translate("abcde","123","cbade"," +") → 321 + +
say translate("low") → LOW
say translate("0110","10","01") → 1001
```

TRIM()

Usage: **TRIM**(*string*)

Removes trailing blanks from the *string* argument.

For example:

```
say length(trim(' abc ')) → 4
```

TRUNC()

Usage: **TRUNC**(*number*,*[places]*)

Returns the integer part of the number argument followed by the specified number of decimal places. The default number of decimal places is 0 and the number is padded with zeroes as necessary.

For example:

```
say trunc(123.456) → 123
say trunc(123.456,4) → 123.4560
```

UPPER()

Usage: **UPPER**(*string*)

Translates the *string* to uppercase. The action of this function is equivalent to that of **TRANSLATE**(*string*), but it is slightly faster for short strings.

For example:

```
say upper('One Fine Day') → ONE FINE DAY
```


VALUE()

Usage: VALUE(*name*)

Returns the value of the symbol represented by the *name* argument.

For example:

```
/* Assume that J has the value 12                               */  
say value('j') → 12
```

VERIFY()

Usage: VERIFY(*string*,*list*,['Match'])

If the **Match** argument is omitted, the function returns the index of the first character in the *string* argument which is **not** contained in the *list* argument or 0 if all of the characters are in the list. If the **Match** keyword is supplied, the function returns the index of the first character which **is** in the list or 0 if none of the characters are.

For example:

```
say verify('123456', '0123456789') → 0  
say verify('123a56', '0123456789') → 4  
say verify('123a45', 'abcdefghij', 'm') → 4
```

WORD()

Usage: WORD(*string*,*n*)

Returns the *n*th word in the *string* argument or the null string if there are fewer than *n* words.

For example:

```
say word('Now is the time ', 2) → is
```

WORDINDEX()

Usage: **WORDINDEX**(*string*,*n*)

Returns the starting position of the *n*th word in the argument string or 0 if there are fewer than *n* words.

For example:

say wordindex('Now is the time ',3) → 8

WORDLENGTH()

Usage: **WORDLENGTH**(*string*,*n*)

Returns the length of the *n*th word in the *string* argument.

For example:

say wordlength('one two three',3)→ 5

WORDS()

Usage: **WORDS**(*string*)

Returns the number of words in the *string* argument.

For example:

say words("'You don't say!'") → 3

WRITECH()

Usage: **WRITECH**(*file*,*string*)

Writes the *string* argument to the given logical file. The returned value is the actual number of characters written.

For example:

say writtech('output', 'Testing') → 7

WRITELN()

Usage: **WRITELN**(*file*,*string*)

Writes the *string* argument to the given logical file with a "newline" appended. The returned value is the actual number of characters written.

For example:

```
say writeln('output', 'Testing')    → 8
```

X2C()

Usage: **X2C**(*string*)

Converts a string of hex digits into the (packed) character representation. Blank characters are permitted in the argument string at byte boundaries.

For example:

```
say x2c('12ab')           → '12ab'x
say x2c('12 ab')          → '12ab'x
say x2c(61)                → a
```

X2D()

Usage: **X2D**(*hex*,*digits*)

Converts a hexadecimal number to decimal.

For example:

```
say x2d('1f')             → 31
```

XRANGE()

Usage: **XRANGE**(*[start]*,*[end]*)

Generates a string consisting of all characters numerically between the specified *start* and *end* values. The default start character is '00'x, and the default end character is 'FF'x. Only the first character of the *start* and *end* arguments is significant.

For example:

```
say xrange()      → '00010203 . . . FDFEFF'x
say xrange('a','f') → 'abcdef'
say xrange('0A'x) → '000102030405060708090A'x
```

Built-In Functions — Examples

The following example programs illustrate many of the Built-In functions that manipulate character strings.

```
/* File name: changestrings.rexx */
/* This AREXX program shows the effect of built-in functions
   that change strings. The functions come in two groups, one that is
   concerned with manipulating individual characters and one that is
   more concerned with manipulating whole strings. */

teststring1 = " every good boy does fine "

/* The first group is composed of the functions strip(), compress(),
   space(), trim(), translate(), delstr(), delword(), insert(), overlay(),
   and reverse(). */

/* Strip() removes only leading and trailing characters. */

/* Print the original string, for comparison. We put a period at
   the end of the string, so you can see what happens
   to the spaces at the end of the string. */
say " every good boy does fine "

/* The same string with its leading and trailing spaces stripped off */
say strip( " every good boy does fine " )"."

/* A failed attempt to remove the leading and trailing "e"'s */
say strip( " every good boy does fine ", "e" )"."

/* The "e"'s were protected by the leading and trailing spaces.
   Removing them exposes the "e"'s to the effects of strip() */
say strip( "every good boy does fine", "e" )"."
```



```

/* Remove "e"'s and spaces from the original string */
say strip( " every good boy does fine ", " e" )"."

/* We switch to using the variable "teststring1", defined above.
Remove only the trailing spaces in the test string. */
say strip( teststring1, T )"."

/* Remove the trailing spaces and the "e" */
say strip( teststring1,T," e" )"."

/* Compress() removes characters anywhere in the string. */
/* This removes all blanks from the test string, "teststring1". */
say compress( teststring1 )
Call time('r')
say time('Civil')                                /* Civilian time
                                                HH:MM{AM I PM} */
say time('h')                                    /* hours since midnight */
say time('m')                                    /* minutes since midnight */
say time('s')                                    /* minutes since midnight */
say time('e')                                    /* elapsed time since reset */

/* Function: TRACE Usage: TRACE([option]) */
say trace()
say trace(trace())                                /* leave it unchanged */

/* Function: TRANSLATE
Usage: TRANSLATE(string,[output],[input],[pad]) */
say translate('aBCdef')                          /* translate to UPPERcase */
say translate('abcdef','1234')
say translate('654321','abcdef','123456')
say translate('abcdef','123','abcdef','+')

/* Function: TRIM Usage: TRIM(string) */
say trim(' abc ')

/* Function: TRUNC Usage: TRUNC(number,[places]) */
say trunc(123.456)
say '$'trunc(134566.123,2)

/* Function: UPPER Usage: UPPER(string) */
say upper('aBCdef12')

```

```
/* Function: VALUE Usage: VALUE(name) */
abc = 'my name'
say value('abc')

/* Function: VERIFY Usage: VERIFY(string,list,['M']) */
say verify('123a45','0123456789')
say verify('abc3de','012456789','M')

/* Function: WORD Usage: WORD(string,n) */
say word('Now is the time',3)

/* Function: WORDINDEX Usage: WORDINDEX(string,n) */
say wordindex('Now is the time ',3)

/* Function: WORDLENGTH Usage: WORDLENGTH(string,n) */
say wordlength('Now is the time ',4)

/* Function: WORDS Usage: WORDS(string) */
say words('Now is the time')

/* Function: WRITECH Usage: WRITECH(logical,string) */
if open('test','ram:test$$','W') then do
    say writech('test','message') /* write the string */
    call close 'test'
end

/* Function: WRITELN Usage: WRITELN(logical,string) */
if open('test','ram:test$$','W') then do
    say writeln('test','message')
    /* write the string (w/newline) */
    call close 'test'
end

/* Function: X2C Usage: X2C(hexstring) */
say x2c('616263') /* convert to character (pack) */

/* Function: XRANGE Usage:
XRANGE([start],[end]) */
say c2x(xrange('f0'x))
say xrange('a','g')

exit
```

Here is the output:

```
every good boy does fine .
every good boy does fine.
every good boy does fine .
very good boy does fin.
very good boy does fin.
every good boy does fine.
every good boy does fin.
everygoodboydoesfine
1:23PM
13
803
48199
0.80
N
N
ABCDEF
abcdef
fedcba
123 + + +
abc
123
$134566.12
ABCDEF12
my name
4
0
the
8
4
4
7
8
abc
F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF
abcdefg
```

REXXSupport.Library Functions

The functions listed in this section are part of the REXXSupport.library. They may only be used if this library has been opened. Below is an example that shows you how to open this library.

```
/* Add rexxsupport.library if it isn't already open. */
if -show ('L', "rexxsupport.library") then do
  /* If the library isn't open, try to open it. */
  if addlib ('rexxsupport.library', 0, -30,0) then
    say "Added rexxsupport.library."
  else do
    say "Rexxsupport.library not available, exiting . . ."
    exit 10 /* exit if addlib failed */
  end
end
```

ALLOCMEM()

Usage: **ALLOCMEM**(*length*,*[attribute]*)

Allocates a block of memory of the specified length from the system free-memory pool and returns its address as a 4-byte string. The optional *attribute* parameter must be a standard EXEC memory allocation flag, supplied as a 4-byte string. The default attribute is for "PUBLIC" memory (not cleared). See the Rom Kernal Manuals for information on memory types and attribute parameters.

This function should be used whenever memory is allocated for use by external programs. It is the user's responsibility to release the memory space when it is no longer needed.

See also **FREEMEM()**.

For example:

```
say c2x(allocmem(1000))      → 00050000
say c2x(allocmem (1000,'00 01 00 01'X ))
                             → 00228400
/* 1000 bytes of CLEAR Public memory */
```


CLOSEPORT()

Usage: **CLOSEPORT**(*name*)

Closes the message port specified by the *name* argument, which must have been allocated by a call to **OPENPORT**() within the current AREXX program. Any messages received but not yet **REPLYed** are automatically returned with the return code set to 10. See also **OPENPORT**().

For example:

```
call closeport myport
```

FREEMEM()

Usage: **FREEMEM**(*address,length*)

Releases a block of memory of the given length to the system freelist. The *address* parameter is a four-byte string, typically obtained by a prior call to **ALLOCMEM**(). **FREEMEM**() cannot be used to release memory allocated using **GETSPACE**(), the AREXX internal memory allocator. The returned value is a boolean success flag. See also **ALLOCMEM**().

For example:

```
MemoryRequest = 1024
MyMem = AllocMem(MemoryRequest)
say c2x(MyMem)                      → 07C987B0
say FreeMem(MyMem, MemoryRequest)   → 1
/* Or: say FreeMem('07C987B0'x, 1024) */
```



Before your program terminates, you **MUST** use a matching **FREEMEM**() to release the exact amount of memory you allocated with each **ALLOCMEM**(). Otherwise you may crash the system or leave memory unavailable until you reboot.

GETARG()

Usage: **GETARG(packet,[n])**

Extracts a command, function name, or argument string from a message packet. The *packet* argument must be a 4-byte address obtained from a prior call to **GETPKT()**. The optional *n* argument specifies the slot containing the string to be extracted and must be less than or equal to the actual argument count for the packet. Commands and function names are always in slot 0; function packets may have argument strings in slots 1-15.

For example:

```
command = getarg(packet)
function = getarg(packet,0)      /* name string */
arg1 = getarg(packet,1)         /* 1st argument */
```

GETPKT()

Usage: **GETPKT(name)**

Checks the message port specified by the *name* argument to see whether any messages are available. The named message port must have been opened by a prior call to **OPENPORT()** within the current AREXX program. The returned value is the 4-byte address of the first message packet or '0000 0000'x if no packets were available.

The function returns immediately whether or not a packet is enqueued at the message port. Programs should never be designed to "busy-loop" on a message port. If there is no useful work to be done until the next message packet arrives, the program should call **WAITPKT()** and allow other tasks to proceed. See also **WAITPKT()**.

For example:

```
packet = getpkt('MyPort')
```

OPENPORT()*Usage:* **OPENPORT**(*name*)

Creates a public message port with the given name. The returned boolean value indicates whether the port was successfully opened. An initialization failure will occur if another port of the same name already exists or if a signal bit couldn't be allocated.

The message port is allocated as a Port Resource node and is linked into the program's global data structure. Ports are automatically closed when the program exits and any pending messages are returned to the sender. See also **CLOSEPORT()**.

For example:

```
success = openport("MyPort")
```

REPLY()*Usage:* **REPLY**(*packet,rc*)

Returns a message packet to the sender, with the primary result field set to the value given by the *rc* argument. The secondary result is cleared. The *packet* argument must be supplied as a 4-byte address and the *rc* argument must be a whole number.

For example:

```
call reply packet,10 /* error return */
```

SHOWDIR()*Usage:* **SHOWDIR**(*directory*,['ALL' | 'File' | 'Dir'],[*pad*])

Returns the contents of the specified directory as a string of names separated by blanks. The second parameter is an option keyword that selects whether all entries, only files, or only subdirectories, will be included.

For example:

```
say showdir('sys:rexx', 'f', ';')  
→ WaitForPort;TS;TE;TCO;RXSET;RXLIB;RXC;RX;HI
```

SHOWLIST()

Usage:

SHOWLIST({'A' | 'D' | 'H' | 'I' | 'L' | 'M' | 'P' | 'R' | 'S' | 'T' | 'V' | 'W'},[*name*],[*pad*])

The first argument selects from the following list: Assigned directories, Device drivers, Handlers, Interrupts, Libraries, Memory list items, Ports, Resources, Semaphores, ready Tasks, Volume names, and Waiting tasks. If only one argument is supplied, Showlist returns a string separated by blanks. If a pad character is supplied, names will be separated by the pad rather than by blanks. If the name parameter is supplied, Showlist returns a boolean value which indicates if the specified list contains that name. Note that names are case-sensitive.

To provide an accurate snapshot of the current list, task switching is forbidden when the list is scanned.

For example:

```
say showlist('P')           → REXX MyCon
say showlist('P',',;')      → REXX;MyCon
say showlist('P','REXX')    → 1
```

STATEF()

Usage: **STATEF**(*filename*)

Returns a string containing information about an external file. The string is formatted as:

```
"{DIR | FILE} length blocks protection days minutes ticks comment."
```

The *length* token gives the file length in bytes, and the *block* token specifies the file length in blocks.

For example:

```
say statef('libs:rexsupport.library')
/* might give "File 2524 5 ----RW-D 4866 817 2088 */
say date('n', 4866)           → 29 Apr 1991
```


WAITPKT()

Usage: **WAITPKT**(*name*)

Waits for a message to be received at the specified (named) port, which must have been opened by a call to **OPENPORT()** within the current AREXX program. The returned boolean value indicates whether a message packet is available at the port. Normally the returned value will be 1 (**True**), since the function waits until an event occurs at the message port.

The packet must then be removed by a call to **GETPKT()** and should be returned eventually using the **REPLY()** function. Any message packets received but not returned when an AREXX program exits are automatically **REPLYed** with the return code set to 10.

For example:

```
call waitpkt 'MyPort'                                /* wait awhile */
```

Tracing and Interrupts

As an added feature, AREXX provides tracing and source-level debugging facilities that are not usually found in a high-level language. *Tracing* refers to the ability to display selected statements in a program as the program executes. When a clause is traced, its line number, source text and related information are displayed on the console. The tracing action of the interpreter is determined by a *trace option* that selects which source clauses will be traced and two modifier flags that control *command inhibition* and *interactive tracing*.

The internal *interrupt system* enables an AREXX program to detect certain synchronous or asynchronous events and to take special actions when they occur. Events such as a syntax error or an external halt request that would normally cause the program to exit can instead be trapped so that corrective actions can be taken.

Tracing Options

Trace options are sometimes called alphabetic options, since the keywords that select an option can be shortened to one letter for convenience. The alphabetic options are:

- **ALL.** All clauses are traced.
- **BACKGROUND.** No tracing is performed and the program cannot be forced into interactive tracing.
- **COMMANDS.** All command clauses are traced before being sent to the external host. Non-zero return codes are displayed on the console.
- **ERRORS.** Commands that generate a non-zero return code are traced after the clause is executed.
- **INTERMEDIATES.** All clauses are traced and intermediate results are displayed during expression evaluation. These include the values retrieved for variables, expanded compound names and the results of function calls.
- **LABELS.** All label clauses are traced as they are executed. A label will be displayed each time a transfer of control takes place.

- **NORMAL.** Command clauses with return codes that exceed the current error failure level are traced after execution and an error message is displayed. This is the default trace option.
- **OFF.**
- **RESULTS.** All clauses are traced before execution and the final result of each expression is displayed. Values assigned to variables by **ARG**, **PARSE** or **PULL** instructions are also displayed.
- **SCAN.** This is a special option that traces all clauses and checks for errors, but suppresses the actual execution of the statements. It is helpful as a preliminary screening step for a newly-created program.

The tracing mode can be set using either the **TRACE** instruction or the **TRACE()** Built-In function. The **RESULTS** trace option is recommended for general-purpose testing. Tracing can be selectively disabled from within a program so that previously tested parts of a program can be skipped.

Display Formatting

Each trace line displayed on the console is indented to show the effective control (nesting) level at that clause and is identified by a special three-character code, as shown in the table below. The source for each clause is preceded by its line number in the program. Expression results or intermediates are enclosed in double quotes so that leading and trailing blanks will be apparent.

Tracing Prefix Codes	
Code	Displayed Values
+++	Command or syntax error
>C>	Expanded compound name
>F>	Result of a function call
>L>	Label clause
>O>	Result of a dyadic operation
>P>	Result of a prefix operation
>U>	Uninitialized variable
>V>	Value of a variable
>>>	Expression or template result
>.>	"Placeholder" token value

The Global Tracing Console

The tracing output from an AREXX program usually goes to the standard output stream **STDOUT** and is therefore interleaved with the normal output of the program. Since this may be confusing at times, a global trace console can be opened to display only tracing output using the **tco** command utility. AREXX programs will automatically divert their tracing output to the new window, which is opened as a standard AmigaDOS console. The user can move it and resize it as required.

The tracing console also serves as the input stream for programs during interactive tracing. When a program pauses for tracing input, the input line must be entered at the trace console. Any number of programs may use the tracing console simultaneously, although it is recommended that only one program at a time be traced.

The tracing console can be closed using the **tcc** command. The closing is delayed until all read requests to the console have been satisfied. Only when all of the active programs indicate that they are no longer using the console will it actually be closed.

Tracing Output

The tracing output from a program is always directed to one of two logical streams. The interpreter first checks for a stream named **STDERR** and directs the output there if the stream exists. Otherwise the trace output goes to the standard output stream **STDOUT** and will be interleaved with the normal console output of the program. The **STDERR** and **STDOUT** streams can be opened and closed under program control, so the programmer has complete control over the destination of tracing output.

In some cases a program may not have a predefined output stream. For example, a program invoked from a host application that did not provide input and output streams would not have an output console. To provide a tracing facility for such programs, the resident process can open a special *global tracing console* for use by any active program. When this console opens, the interpreter automatically opens a stream named **STDERR** for each AREXX program in which **STDERR** is not currently defined and the program then diverts its tracing output to the new stream.

The global console can be opened and closed using the command utilities **tco** and **tcc**, respectively. The console may not close immediately upon request, however. The resident process waits until all active programs have diverted their tracing streams back to the default state before actually closing the console.

The trace stream (**STDERR** or **STDOUT**) is also used for trace input, so a program in interactive tracing mode will wait for user input from this console. The global tracing console is always shared among all currently active programs. Since it may be confusing to have several programs simultaneously writing to the same console, it is recommended that only one program at a time be traced using the global console.

Command Inhibition

AREXX provides a tracing mode called *command inhibition* that suppresses host commands. In this mode command clauses are evaluated in the normal manner, but the command is not actually sent to the external host, and the return code is set to zero. This provides a way to test programs that issue potentially destructive commands, such as erasing files or formatting disks. Command inhibition does not apply to command clauses that are entered interactively. These commands are always performed, but the value of the special variable **RC** is left unchanged.

Command inhibition may be used in conjunction with any trace option. It is controlled by the **“!”** character, which may appear by itself or may precede any of the alphabetic options in a **TRACE** instruction. Each occurrence of the **“!”** character “toggles” the inhibition mode currently in effect. Command inhibition is cleared when tracing is set to **OFF**.

Interactive Tracing

Interactive tracing is a debugging facility that allows the user to enter source statements while a program is executing. These statements may be used to examine or modify variable values, issue commands or otherwise interact with the program. Any valid language statements can be entered interactively, with the same rules and restrictions that apply to the **INTERPRET** instruction. In particular, compound statements such as **DO** and **SELECT** must be complete within the entered line.

Interactive tracing can be used with any of the trace options. While in interactive tracing mode, the interpreter pauses after each traced clause and prompts for input with the code `"+++ "`. At each pause, three types of user responses are possible:

- If a null line is entered, the program continues to the next pause point.
- If a `"="` character is entered, the preceding clause is executed again.
- Any other input is treated as a debugging statement and is scanned and executed.

The pause points during interactive tracing are determined by the tracing option currently in effect, as the interpreter pauses only after a traced clause. However, certain instructions cannot be safely (or sensibly) re-executed, so the interpreter will not pause after executing one of these. The "no-pause" instructions are **CALL**, **DO**, **ELSE**, **IF**, **THEN** and **OTHERWISE**. The interpreter will also not pause after any clause that generated an execution error.

Interactive tracing mode is controlled by the "?" character, either by itself or in combination with an alphabetic trace option. Any number of "?" characters may precede an option and each occurrence toggles the mode currently in effect. For example, if the current trace options was **NORMAL**, then **"TRACE ?R"** would set the option to **RESULTS** and select interactive tracing mode. A subsequent **"TRACE ?"** would turn off interactive tracing.

Error Processing

The AREXX interpreter provides special error processing while it executes debugging statements. Errors that occur during interactive debugging are reported, but do not cause the program to terminate. This special processing applies only to the statements that were entered interactively. Errors occurring in the program source statements are treated in the usual way whether or not the interpreter is in interactive tracing mode.

In addition to the special error processing, the interpreter also disables the internal interrupt flags during interactive debugging. This is necessary to prevent an accidental transfer of control due to an error or uninitialized variable. However, if a **"SIGNAL label"** instruction is entered, the transfer will take place and any remaining interactive input will be abandoned. The **SIGNAL** instruction can still be used to alter the interrupt flags and the new settings will take effect when the interpreter returns to normal processing.

Failure Level for Commands

Each AREXX task initializes its command failure level to the client's failure level (usually 10) and this level controls reporting of command errors. This will help suppress printing of nuisance command errors. The failure level can be changed using `OPTIONS FAILAT` as before. Command errors (`RC > 0`) and failures (`RC >= failat`) can be separately trapped using `SIGNAL ON ERROR` and `SIGNAL ON FAILURE`.

The External Tracing Flag

The AREXX resident process maintains an *external tracing flag* that can be used to force programs into interactive tracing mode. The tracing flag can be set using the `ts` command utility. When the flag is set, any program not already in interactive tracing mode will enter it immediately. The internal trace option is set to **RESULTS** unless it is currently set to **INTERMEDIATES** or **SCAN**, in which case it remains unchanged. Programs invoked while the external tracing flag is set will begin executing in interactive tracing mode.

The external tracing flag provides a way to regain control over programs that are caught in loops or are otherwise unresponsive. Once a program enters interactive tracing mode, the user can step through the program statements and diagnose the problem. There is one caveat, though; external tracing is a global flag, so all currently-active programs are affected by it. The tracing flag remains set until it is cleared using the `"te"` command utility. Each program maintains an internal copy of the last state of the tracing flag and sets its tracing option to **OFF** when it observes that the tracing flag has been cleared. Programs in **BACKGROUND** tracing mode do not respond to the external tracing flag.

Interrupts

AREXX maintains an internal interrupt system that can be used to detect and trap certain error conditions. When an interrupt is enabled and its corresponding condition arises, a transfer of control to the label specific to that interrupt occurs. This allows a program to retain control in circumstances that might otherwise cause the program to terminate. The interrupt conditions can be caused by either *synchronous* events like a syntax error or *asynchronous* events like a "control-C" break request. Note that these internal interrupts are completely separate from the hardware interrupt system managed by the EXEC operating system.

The interrupts supported by AREXX are described below. The name assigned to each is actually the label to which control will be transferred. Thus, a **SYNTAX** interrupt will transfer control to the label "**SYNTAX:**". Interrupts can be enabled or disabled using the **SIGNAL** instruction. For example, the instruction "**SIGNAL ON SYNTAX**" would enable the **SYNTAX** interrupt.

- **BREAK_C.** This interrupt will trap (detect and treat as a signal, and not as normal output) a control-C break request generated by DOS. If the interrupt is not enabled, the program terminates immediately with the error message "**Execution halted**" and returns with the error code set to 2.
- **BREAK_D.** The interrupt will trap a control-D break request issued by DOS. The break request is ignored if the interrupt is not enabled.
- **BREAK_E.** The interrupt will trap a control-E break request issued by DOS. The break request is ignored if the interrupt is not enabled.

- **BREAK_F.** The interrupt will trap a control-F break request issued by DOS. The break request is ignored if the interrupt is not enabled.
- **ERROR.** This interrupt is generated by any host command that returns a non-zero code.
- **HALT.** An external halt request will be trapped if this interrupt is enabled. Otherwise, the program terminates immediately with the error message "**Execution halted**" and returns with the error code set to 2.
- **IOERR.** Errors detected by the I/O system will be trapped if this interrupt is enabled.
- **NOVALUE.** An interrupt will occur if an uninitialized variable is used while this condition is enabled. The usage could be within an expression, in the **UPPER** instruction, or with the **VALUE()** built-in function.
- **SYNTAX.** A syntax or execution error will generate this interrupt. Not all such errors can be trapped, however. In particular, certain errors occur before a program is actually executing and those detected by the AREXX external interface, cannot be trapped by the **SYNTAX** interrupt.

When an interrupt forces a transfer of control, all of the currently active control ranges are dismantled and the interrupt that caused the transfer is disabled. This last action is necessary to prevent a possible recursive interrupt loop. Only the control structures in the current environment are affected, so an interrupt generated within a function will not affect the caller's environment.

Special Variables. Two special variables are affected when an interrupt occurs. The variable **SIGL** is always set to the current line number before the transfer of control takes place, so that the program can determine which source line was being executed. When an **ERROR** or **SYNTAX** interrupt occurs, the variable **RC** is set to the error code that caused the condition. For **ERROR** interrupts, this value will be a command return code and can usually be interpreted as an error severity level. The value for **SYNTAX** interrupts is always an AREXX error code.

Interrupts are useful primarily to allow a program to take special error-recovery actions. Such actions might involve informing external programs that an error occurred or simply to report further diagnostics to help in isolating the problem. In the following example, the program issues a "message" command to an external host called "MyEdit" whenever a syntax error is detected:

```
/* A macro program for 'MyEdit' */
signal on syntax                /* enable interrupt */
.
. (normal processing)
.
exit
syntax:                        /* syntax error detected */
    address 'MyEdit'
    'message' 'error' rc errortext(rc)
    exit 10
```


Parsing and Templates

Parsing is an operation that extracts substrings from a string and assigns them to variables. It corresponds roughly to the notion of a “formatted read” used in other languages, but has been generalized in several ways. Parsing is performed using the **PARSE** instruction or its variants **ARG** and **PULL**. The input for the operation is called the *parse string* and can come from several sources.

Parsing is controlled by a *template*, a group of tokens that specifies both the variables to be given values and the way to determine the value strings. Templates were described briefly with the **PARSE** instruction. A more formal description of their structure and operation is described below.

String-manipulation functions like **SUBSTR()** and **INDEX()** could also be used for parsing, but it is more efficient to use the instruction statements. This is especially true if many fields are to be extracted from a string.

Template Structure

The tokens that are valid in a template are symbols, string, operators, parentheses and commas. Any blanks that may be present as separators are removed before the template is processed. The tokens in a template ultimately serve to specify one of the two basic template objects:

- *Markers* determine a scan position within the parse string.
- *Targets* are symbols to be assigned a value.

With these objects in mind, the parsing process can be described as one of associating with each target a starting and ending position in the parse string. The substring between these positions then becomes the value for the target.

Markers. There are three types of marker objects:

- *Absolute* markers specify an actual index position in the parse string.
- *Relative* markers specify a positive or negative offset from the current position.
- *Pattern* markers specify a position implicitly, by matching the pattern against the parse string beginning at the current scan position.

Targets. Targets are usually specified by variable symbols. The *placeholder* is a special type of target and is denoted by a period (.) symbol. A placeholder behaves like a normal target except that a value is not actually assigned to it.

Targets, like markers, can affect the scan position if value strings are being extracted by *tokenization*. Parsing by tokenization extracts words (tokens) from the parse string and is used whenever a target is followed immediately by another target. During tokenization the current scan position is advanced past any blanks to the start of the next word. The ending index is the position just past the end of the word, so that the value string has neither leading nor trailing blanks.

Template Objects

Each template object is specified by one or more tokens, which have the following interpretations.

Symbols. A symbol token may specify either a target or a marker object. If it follows an operator token (+, - or =) it represent a marker and the symbol value is used as an absolute or relative position. Symbols enclosed in parentheses specify pattern markers and the symbol value is used as the pattern string.

If neither of the preceding cases applies and the symbol is a variable, then it specifies a target. Fixed symbols always specify absolute markers and must be whole numbers, except for the period (.) symbol, which defines a placeholder target.

Strings. A string token always represents a pattern marker.

Parentheses. A symbol enclosed in parentheses is a pattern marker and the value of the symbol is used as the pattern string. While the symbol may be either fixed or variable, it will usually be a variable, since a fixed pattern could be given more simply as a string.

Operators. The three operators (+, - and =) are valid within a template and must be followed by a fixed or variable symbol. The value of the symbol is used as a marker and must therefore represent a whole number. The "+" and "-" operators signify a relative marker, whose value is negated by the "-" operator. The "=" operator indicates an absolute marker and is optional if the marker is defined by a fixed symbol.

Commas. The comma (,) marks the end of a template and is used as a separator when multiple templates are provided with an instruction. The interpreter obtains a new parse string before processing each succeeding template. For some source

options, the new string will be identical to the previous one. The **ARG**, **EXTERNAL** and **PULL** options will generally supply a different string, as will the **VAR** option if the variable has been modified.

The Scanning Process

Scan positions are expressed as an index in the parse string and can range from 1 (the start of the string) to the length of the string plus 1 (the end). An attempt to set the scan position before the start or after the end of the string instead sets it to the beginning or end, respectively.

The substring specified by two scan indices includes the characters from the starting position up to, but not including, the ending position. For example, the indices 1 and 10 specify characters 1-9 in the parse string. One additional rule is applied if the second scan index is less than or equal to the first; the *remainder* of the parse string is used as the substring. This means that a template specification like:

parse arg 1 all 1 first second

will assign the entire parse string to the variable **ALL**. Of course, if the current scan index is already at the end of the parse string, then the remainder is just the null string.

When a pattern marker is matched against the parse string, the marker position is the index of the first character of the matched pattern or the end of the string if no match was found. The pattern is removed from the string whenever a match is found. This is the only operation that modifies the parse string during the parsing process.

Templates are scanned from left to right with the initial scan index set to 1, the start of the parse string. The scan position is updated each time a marker object is encountered, according to the type and value of the marker. Whenever a target object is found, the value to be assigned is determined by examining the next template object. If the next object is another target, the value string is determined by tokenizing the parse string. Otherwise, the current scan position is used as the start of the value string and the position specified by the following marker is used as the end point.

The scan continues until all of the objects in the template have been used. Note that every target will be assigned a value. Once the parse string has been exhausted, the null string is assigned to any remaining targets.

Templates in Action

The following section provides some examples of parsing with templates.

Parsing by Tokenization

Computer programs frequently require splitting a string into its component words or tokens. This is easily accomplished with a template consisting entirely of variables (targets).

```
/* Assume "hammer 1 each $600.00" was entered */  
pull item qty units cost .
```

In this example the input line from the **PULL** instruction is split into words and assigned to the variables in the template. The variable **item** receives the value "**hammer**", **qty** is set to "**1**", **units** is set to "**each**" and **cost** gets the value "**\$600.00**". The final placeholder (.) is given a null value, since there are only four words in the input. However, it forces the preceding

variable **cost** to be given a tokenized value. If the placeholder were omitted, the remainder of the parse string would be assigned to **cost**, which would then have a leading blank.

In the next example, the first word of a string is removed and the remainder is placed back in the string. The process continues until no more words are extracted.

```
answer = "Only Amiga makes it possible."  
do forever  
  parse var answer first answer  
  /* place 1st word into 'first' and the rest into 'answer'. */  
  if first == '' then leave  
  /* stop if there are no more words */  
  say answer  
end
```

Here is the output:

```
Amiga makes it possible.  
makes it possible.  
it possible.  
possible.
```

Note the space at the beginning of each line.

Pattern Parsing

The next example uses pattern markers to extract the desired fields. The "pattern" in this case is very simple — just a single character — but in general can be an arbitrary string of any length. This form of parsing is useful whenever delimiter characters are present in the parse string.

```
/* Assume an argument string "12,35.5,1" */  
arg hours ',' rate ',' withhold
```

Keep in mind that the pattern is actually *removed* from the parse string when a match is found. If the parse string is scanned again from the beginning, the length and structure of the string may be different than at the start of the parsing process. However, the original source of the string is never modified.

Positional Markers

Parsing with positional markers is used whenever the files of interest are known to be in certain positions in a string. In the next example, the records being processed contain a variable length field. The starting position and length of the field are given in the first part of the record and a variable positional marker is used to extract the desired field.

```
/* records look like:          */  
/* start: 1-5                 */  
/* length: 6-10               */  
/* name: @ (start,length)     */
```

parse value record with 1 start + 5 length + 5 = start name + length

The "**= start**" sequence in the above example is an absolute marker whose value is the position placed in the variable **start** earlier in the scan. The "**+ length**" sequence supplies the effective length of the field.

Multiple Templates

It is sometimes useful to specify more than one template with an instruction, which can be done by separating the templates with a comma. In this next example, the **ARG** instruction (or **PARSE UPPER ARG**) is used to access the argument strings provided when the program was called. Each template accesses the succeeding argument string.

```
/* Assume arguments are ('one two',12,sort) */
arg first second,amount,action,option
```

The first template consists of the variables **first** and **second**, which are set to the values "one" and "two", respectively. In the next two templates **amount** gets the value "12" and **action** is set to "SORT". The last template consists of the variable "option", which is set to the null string, since only three arguments were available.

When multiple templates are used with the **EXTERNAL** or **PULL** source options, each additional template requests an additional line of input from the user.

In the next example two lines of input are read:

```
/* read last, first, and middle names and ssn */
pull last ',' first middle,ssn
```

The first input line is expected to have three words, the first of which is followed by a comma, which are assigned to the variables **last**, **first** and **middle**. The entire second input line is assigned to the variable **ssn**.

Multiple templates can be useful even with a source option that returns the identical parse string. If the first template included pattern markers that altered the parse string, the subsequent templates could still access the original string. Note that subsequent parse strings obtained from the **VALUE** source do not cause the expression to be reevaluated, but only retrieve the prior result.

Additional Notes

- The AREXX interface command parser has been generalized to recognize double-delimiter sequences within a (quoted) string file. The quoting convention that allows you to enter a REXX program as a string is very convenient for short programs, but it was easy to run out of quoting levels in longer programs. Note that single and double-quotes *within* a REXX program are exactly equivalent, but that the external environment may make a distinction. For example, AmigaDOS uses only the double-quote as its quoting character, so string files entered from a CLI must begin with a double-quote, at least if you wish to include any semicolons. For example:

```
rx "say 'It's possible, indeed; you ain't seen nothin' yet!'"
→ It's possible, indeed; you ain't seen nothin' yet!
```

```
rx "say ' ' 'Hello!' ' ' '"
→ "Hello!"
```

REXXC Directory

AREXX is supplied with a number of command utilities, located in the REXXC Directory, that provide various control functions. These are executable modules that can be run from the Shell and normally reside in the system **rexxc** directory for convenience. These programs are relevant only when the AREXX resident process is active.

HI

Usage: **HI**

Sets the global halt flag, which causes all active AREXX programs to receive an external halt request. Each program will exit immediately unless its **HALT** interrupt has been enabled. The halt flag does not remain set, but is cleared automatically after all current programs have received the request.

RX

Usage: **RX** *name [arguments]*

This command launches an AREXX program. If the specified *name* includes an explicit path, only that directory is searched for the program; otherwise, the current directory and **REXX**: are checked for a program with the given name. The optional argument string is passed to the program.

RXSET

Usage: **RXSET** [*name* *[[=]* *value*]

Adds a (name,value) pair to the Clip List. Name strings are assumed to be in mixed case. If a pair with the same name already exists, its value is replaced with the current string. If a name without a value string is given, the entry is removed from the Clip List. If RXSET is invoked without arguments, it will list all (name, value) pairs in the Clip List.

RXC

Usage: **RXC**

Closes the resident process. The "REXX" public port is withdrawn immediately and the resident process exits as soon as the last AREXX program finishes. No new programs can be launched after a "close" request.

TCC

Usage: **TCC**

Closes the global tracing console as soon as all active programs are no longer using it. All read requests queued to the console must be satisfied before it can be closed.

TCO

Usage: **TCO**

Opens the global tracing console. The tracing output from all active programs is diverted automatically to the new console. The console window can be moved and resized by the user and can be closed with the "TCC" command.

TE

Usage: **TE**

Clears the global tracing flag, which forces the tracing mode to **OFF** for all active AREXX programs.

TS

Usage: **TS**

Starts interactive tracing by setting the external trace flag, which forces all active AREXX programs into interactive tracing mode. Programs will start producing trace output and will pause after the next statement. This command is useful

for regaining control over programs caught in infinite loops or otherwise misbehaving. The trace flag remains set until cleared by the **TE** command, so subsequent program invocations will begin executing in interactive tracing mode.

WaitForPort

Usage: **WaitForPort** [name of port]

WaitForPort is a command utility that specifies a port and then waits 10 seconds for the port to appear. A return code of 0 indicates that the port was found. A return code of 5 indicates that the application is not currently running or that the port itself does not exist. Note that port names are case sensitive.

For example:

```
WaitForPort ED_1  
WaitForPort MyPort
```

Error Messages

When the AREXX interpreter detects an error in a program, it returns an error code to indicate the nature of the problem. Errors are normally handled by displaying the error code, the source line number where the error occurred and a brief message explaining the error condition. Unless the **SYNTAX** interrupt has been previously enabled (using the **SIGNAL** instruction), the program then terminates and control returns to the caller. Most syntax and execution errors can be trapped by the **SYNTAX** interrupt, allowing the user to retain control and perform whatever special error processing is required. Certain errors are generated outside of the context of an

AREXX program and therefore cannot be trapped by this mechanism. Refer to Tracing and Interrupts for further information on error trapping and processing.

Associated with each error code is a *severity level* that is reported to the calling program as the primary result code. The error code itself is returned as the secondary result. The subsequent propagation or reporting of these codes is of course dependent on the external (calling) program.

The following pages list all of the currently-defined error codes, along with the associated severity level and message string.

The standards are:

- 5 – least serious
- 10 – moderately serious
- 20 – most serious

Error: 1 Severity: 5 Message: Program not found

The named program could not be found or was not an AREXX program. AREXX programs are expected to start with a `"/"` sequence. This error is detected by the external interface and cannot be trapped by the **SYNTAX** interrupt.

Error: 2 Severity: 10 Message: Execution halted

A control-C break or an external halt request was received and the program terminated. This error will be trapped if the **HALT** interrupt has been enabled.

Error: 3 Severity: 20 Message: Insufficient memory

The interpreter was unable to allocate enough memory for an operation. Since memory space is required for all parsing and execution operations, this error cannot usually be trapped by the **SYNTAX** interrupt.

Error: 4 Severity: 10 Message: Invalid character

A non-ASCII character was found in the program. Control codes and other non-ASCII characters may be used in a program by defining them as hex or binary strings. This is a scan-phase error and cannot be trapped by the SYNTAX interrupt.

Error: 5 Severity: 10 Message: Unmatched quote

A closing single or double quote was missing. Check that each string is properly delimited. This is a scan-phase error and cannot be trapped by the SYNTAX interrupt.

Error: 6 Severity: 10 Message: Unterminated comment

The closing `"*/"` for a comment field was not found. Remember that comments may be nested, so each `"/*"` must be matched by a `"*/"`. This is a scan-phase error and cannot be trapped by the SYNTAX interrupt.

Error: 7 Severity: 10 Message: Clause too long

A clause was too long for the internal buffer. The source line in question should be broken into smaller parts. This is a scan-phase error and cannot be trapped by the SYNTAX interrupt.

Error: 8 Severity: 10 Message: Invalid token

An unrecognized lexical token was found or a clause could not be properly classified. This is a scan-phase error and cannot be trapped by the SYNTAX interrupt.

Error: 9 Severity: 10 Message: Symbol or string too long

An attempt was made to create a string longer than the maximum allowed by the interpreter.

Error: 10 Severity: 10 Message: Invalid message packet

An invalid action code was found in a message packet sent to the AREXX resident process. The packet was returned without being processed. This error is detected by the external interface and cannot be trapped by the SYNTAX interrupt.

Error: 11 Severity: 10 Message: Command string error

A command string could not be processed. This error is detected by the external interface and cannot be trapped by the SYNTAX interrupt.

Error 12: Severity: 10 Message: Error return from function

An external function returned a non-zero error code. Check that the correct parameters were supplied to the function.

Error 13: Severity: 10 Message: Host environment not found

The message port corresponding to a host address string could not be found. Check that the required external host is active.

Error 14: Severity: 10 Message: Requested library not found

An attempt was made to open a function library included in the Library List, but the library could not be opened. Check that the correct name and version of the library were specified when the library was added to the resource list.

Error 15: Severity: 10 Message: Function not found

A function was called that could not be found in any of the currently accessible libraries and could not be located as an external program. Check that the appropriate function libraries have been added to the Libraries List.

Error: 16 Severity: 10 Message: Function did not return value

A function was called which failed to return a result string, but did not otherwise report an error. Check that the function was programmed correctly or invoke it using the CALL instruction.

Error 17: Severity: 10 Message: Wrong number of arguments

A call was made to a function which expected more (or fewer) arguments. This error will be generated if a built-in or external function is called with more arguments than can be accommodated in the message packet used for external communications.

Error 18: Severity: 10 Message: Invalid argument to function

An inappropriate argument was supplied to a function or a required argument was missing. Check the parameter requirements specified for the function.

Error 19: Severity: 10 Message: Invalid PROCEDURE

A **PROCEDURE** instruction was issued in an invalid context. Either no internal functions were active or a **PROCEDURE** had already been issued in the current storage environment.

Error 20: Severity: 10 Message: Unexpected THEN or WHEN

A **WHEN** or **THEN** instruction was executed outside of a valid context. The **WHEN** instruction is valid only within a **SELECT** range and **THEN** must be the next instruction following an **IF** or **WHEN**.

Error 21: Severity: 10 Message: Unexpected ELSE or OTHERWISE

An **ELSE** or **OTHERWISE** was found outside of a valid context. The **OTHERWISE** instruction is valid only within a **SELECT** range. **ELSE** is valid only following the **THEN** branch of an **IF** range.

Error 22: Severity: 10 Message: Unexpected BREAK, LEAVE or ITERATE

The **BREAK** instruction is valid only within a **DO** range or inside an **INTERPRET**ed string. The **LEAVE** and **ITERATE** instructions are valid only within an *iterative* **DO** range.

Error 23: Severity: 10 Message: Invalid statement in SELECT

An invalid statement was encountered within a **SELECT** range. Only **WHEN**, **THEN** and **OTHERWISE** statements are valid within a **SELECT** range, except for the conditional statements following **THEN** or **OTHERWISE** clauses.

Error 24: Severity: 10 Message: Missing or multiple THEN

An expected **THEN** clause was not found or another **THEN** was found after one had already been executed.

Error 25: Severity: 10 Message: Missing OTHERWISE

None of the **WHEN** clauses in a **SELECT** succeeded, but no **OTHERWISE** clause was supplied.

Error 26: Severity: 10 Message: Missing or unexpected END

The program source ended before an **END** was found for a **DO** or **SELECT** instruction or an **END** was encountered outside a **DO** or **SELECT** range.

Error: 27 Severity: 10 Message: Symbol mismatch

The symbol specified on an **END**, **ITERATE** or **LEAVE** instruction did not match the index variable for the associated **DO** range. Check that the active loops have been nested properly.

Error: 28 Severity: 10 Message: Invalid DO syntax

An invalid **DO** instruction was executed. An initializer expression must be given if a **TO** or **BY** expression is specified and a **FOR** expression must yield a non-negative integer result.

Error: 29 Severity: 10 Message: Incomplete IF or SELECT

An **IF** or **SELECT** range ended before all of the required statements were found. Check whether the conditional statement following a **THEN**, **ELSE** or **OTHERWISE** clause was omitted.

Error: 30 Severity: 10 Message: Label not found

A label specified by a **SIGNAL** instruction or implicitly referenced by a enabled interrupt could not be found in the program source. Labels defined dynamically by an **INTERPRET** instruction or by interactive input are not included in the search.

Error: 31 Severity: 10 Message: Symbol expected

A non-symbol token was found where only a symbol token is valid. The **DROP**, **END**, **LEAVE**, **ITERATE** and **UPPER** instructions may only be followed by a symbol token and will generate this error if anything else is supplied. This message will also be issued if a required symbol is missing.

Error: 32 Severity: 10 Message: Symbol or string expected

An invalid token was found in a context where only a symbol or string is valid.

Error: 33 Severity: 10 Message: Invalid keyword

A symbol token in an instruction clause was identified as a keyword, but was invalid in the specific context.

Error: 34 Severity: 10 Message: Required keyword missing

An instruction clause required a specific keyword token to be present, but it was not supplied. For example, this message will be issued if a **SIGNAL ON** instruction is not followed by one of the interrupt keywords (e.g. **SYNTAX**).

Error: 35 Severity: 10 Message: Extraneous characters

A seemingly valid statement was executed, but extra characters were found at the end of the clause.

Error: 36 Severity: 10 Message: Keyword conflict

Two mutually exclusive keywords were included in an instruction clause or a keyword was included twice in the same instruction.

Error: 37 Severity: 10 Message: Invalid template

The template provided with an **ARG**, **PARSE** or **PULL** instruction was not properly constructed.

Error: 38 Severity: 10 Message: Invalid TRACE request

The alphabetic keyword supplied with a **TRACE** instruction or as the argument to the **TRACE()** built-in function was not valid.

Error: 39 Severity: 10 Message: Uninitialized variable

An attempt was made to use an uninitialized variable while the **NOVALUE** interrupt was enabled.

Error: 40 Severity: 10 Message: Invalid variable name

An attempt was made to assign a value to a fixed symbol.

Error: 41 Severity: 10 Message: Invalid expression

An error was detected during the evaluation of an expression. Check that each operator has the correct number of operands and that no extraneous tokens appear in the expression. This error will be detected only in expressions that are actually evaluated. No checking is performed on expressions in clauses that are being skipped.

Error: 42 Severity: 10 Message: Unbalanced parentheses

An expression was found with an unequal number of opening and closing parentheses.

Error: 43 Severity: 10 Message: Nesting limit exceeded

The number of subexpressions in an expression was greater than the maximum allowed. The expression should be simplified by breaking it into two or more intermediate expressions.

Error: 44 Severity: 10 Message: Invalid expression result

The result of an expression was not valid within its context. For example, this message will be issued if an increment or limit expression in a **DO** instruction yields a non-numeric result.

Error: 45 Severity: 10 Message: Expression required

An expression was omitted in a context where one is required. For example, the **SIGNAL** instruction, if not followed by the keywords **ON** or **OFF**, must be followed by an expression.

Error: 46 Severity: 10 Message: Boolean value not 0 or 1

An expression result was expected to yield a boolean result, but evaluated to something other than 0 or 1.

Error: 47 Severity: 10 Message: Arithmetic conversion error

A non-numeric operand was used in an operation requiring numeric operands. This message will also be generated by an invalid hex or binary string.

Error: 48 Severity: 10 Message: Invalid operand

An operand was not valid for the intended operation. This message will be generated if an attempt is made to divide by 0 or if a fractional exponent is used in an exponentiation operation.

Limits

Language definitions seldom include predefined limits to the program structures that can be created. Only a few such restrictions were imposed in implementing AREXX and most of the internal structures are limited only by the total amount of memory available. The current implementation limits are listed below.

- *Length of Strings.* Strings, symbol names and value strings are limited to a maximum length of 65,535 bytes.
- *Arguments to Functions.* Built-In and external functions are limited to a maximum of 15 arguments. There is no limit to the number of arguments that may be passed to an internal function.
- *Subexpression Nesting.* The maximum nesting level for subexpressions is 32.

Compatibility

AREXX departs in a few ways from the REXX language definition. The differences can be classified as omissions or extensions and are described below.

Omissions. A language standard omitted from this implementation is the arbitrary-precision arithmetic facility. Arithmetic operations are performed using IEEE double-precision and limited to about 14 digits of precision.

NOTE: The following instructions **PUSH** and **QUEUE** are included with the System 2.0 software, but are not fully supported. Future software versions will provide full support.

PUSH

Usage: **PUSH** *[expression]*

The **PUSH** instruction is used to prepare a stream of data to be read by a command shell or other program. It appends a "newline" to the result of the expression and then stacks or "pushes" it into the **STDIN** stream. Stacked lines are placed in the stream in "last-in, first-out" order and are then available to be read just as though they had been entered interactively.

For example, after issuing the instructions:

```
push line 1  
push line 2  
push line 3
```

the stream would be read in the order "line 3," "line 2," and "line 1".

There are several restrictions governing the use of the **PUSH** instruction and its alter ego **QUEUE**. These instructions use a special I/O mechanism to accomplish their task and as a result, can be used only with an interactive (stream-model) I/O device like a console or pipe. The stream must be managed with a DOS handler that supports the special **ACTION_STACK** (for **PUSH**) or **ACTION_QUEUE** (for **QUEUE**) command.

PUSH allows the **STDIN** stream to be used as a private scratchpad to prepare data for subsequent processing. For example, several files could be concatenated with delimiters between them by simply reading the input files, **PUSH**ing the line into the stream and inserting a delimiter where required.

QUEUE

Usage: **QUEUE** *[expression]*

The **QUEUE** instruction is used to prepare a stream of data to be read by a command shell or other program. It is very similar to the preceding **PUSH** instruction and differs only in that the data lines are placed in the **STDIN** stream in "first-in, first-out" order. In this case, the instructions:

```
queue line 1  
queue line 2  
queue line 3
```

would be read in the order "line 1", "line 2", and "line 3". The **QUEUED** lines always precede all interactively-entered lines and always follow any **PUSHed** (stacked) lines.

The same restrictions noted with the use of the **PUSH** instruction apply to the **QUEUED** instruction. The queuing mechanism uses the **ACTION_QUEUE** command, so the DOS handler associated with the **STDIN** stream must support this command.

In most cases the choice of whether to use **PUSH** or **QUEUE** is just a matter of convenience or personal preference. Each provides a "scratch pad" facility similar to that provided by an I/O pipe, but useful within one program or task rather than just for interprocess communications.

For example:

```
/* Queue commands for compile and link */  
queue "cc main"  
queue "blink c.o + main.o library amiga.lib to myprog"
```

Extensions. The following extensions to the AREXX language standard have been included in this implementation:

- **BREAK Instruction.** A new instruction called **BREAK** has been implemented. It is used to exit from the scope of any **DO** or **INTERPRET** instruction.
- **ECHO Instruction.** The **ECHO** instruction has been included as a synonym for **SAY**.
- **SHELL Instruction.** The **SHELL** instruction has been included as a synonym for **ADDRESS**.
- **SIGNAL Options.** Several additional **SIGNAL** keywords have been implemented. **BREAK_C**, **BREAK_D**, **BREAK_E** and **BREAK_F** will detect and trap the control-C through Control-F signals passed by AmigaDOS. The **IOERR** keyword traps errors detected by the I/O system.
- **Stem Symbols.** A stem symbol is valid anywhere that a simple symbol could be employed.
- **Template Processing.** Templates have been generalized in several ways. Variable symbols may be used as positional tokens if preceded by an operator; the "=" operator is used to denote an absolute position. Multiple templates can be used with all source forms of the **PARSE** instruction.

Appendix A

Troubleshooting

If you encounter a problem while using the system software, check the following table.

Symptom	Cause	Remedy
Display is flickering; screen is not positioned properly.	The wrong display mode is selected.	Open the ScreenMode editor in the Prefs drawer and reselect the appropriate display mode. (See page 3-37). If the screen display is so bad that you cannot understand it all, you may have to reboot with a different Workbench disk.
A requester asks you to insert a particular volume into any drive.	The system cannot find the program it is looking for. The program may have been moved to a different drawer (if you have a hard disk) or the floppy disk containing the program may have been renamed.	Check the name of the floppy disk or drawer containing the program. If the name is different than what the program is looking for, you may need to add an ASSIGN statement to your User-startup file (see page 6-8).

Symptom	Cause	Remedy
A requester states that there is not enough memory to load a program.	You have too many programs running and there is not enough RAM left to start another program.	Close any unnecessary windows.
A requester states that your disk cannot be validated or that it has a read/write error.	Your disk may have become corrupt.	Use DISKDOCTOR, or another disk repair utility, to try to retrieve your files.
You cannot move the pointer, and keyboard input has no effect.	Your program has crashed.	Reboot.
The screen goes blank, then a flashing red box appears stating an error, such as Not Enough Memory.	A program performed an illegal action which was serious enough to cause a system failure.	Press the left mouse button, and the computer will reboot.
A flashing green box appears stating Recoverable Alert.	A program performed an illegal action which caused an error from which the system can recover.	Press the left mouse button.

Appendix B. Printers

This appendix covers the printer drivers that are included in the Devs/Printers drawer of the Extras2.0 disk as well as the printer escape sequences used by the Amiga.

If you have a hard disk system, the printer drivers will be in the Devs/Printers drawer of your System2.0 partition.



Printer Drivers

A printer driver acts as a translator. The Amiga has one way of encoding information, but different printers require information in different formats. The printer driver takes the information from the Amiga and translates it into the proper format for the particular printer.

The printer drivers listed in this section are:

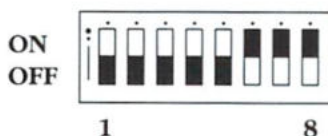
CalComp_ColorMaster	ImagewriterII
CalComp_ColorMaster2	NEC_Pinwriter
CBM_MPS1000	Okidata_293I
Diablo_630	Okidata_92
EpsonQ	Okimate_20
EpsonX	Seiko_5300
EpsonXOld	Seiko_5300a
Howtek_Pixelmaster	Tektronix_4693D
HP_Deskjet	Tektronix_4696
HP_LaserJet	Toshiba_P351C
HP_PaintJet	Toshiba_P351SX
HP_ThinkJet	Xerox_4020

The specifications frequently refer to the gadgets in the Printer and PrinterGfx editors. Different gadget settings can have specific effects on your printout. For instance, you'll notice that many of the drivers support several densities. Density refers to the number of dots per inch used to make the printout. The higher the number of dots, the smaller the dots will be and

the clearer the picture. However, the higher the density, the longer it takes to print. In the case of multiple densities, you must decide whether you prefer faster printing or a higher quality print. You can select the appropriate density with the Density gadget of the PrinterGfx editor.

Densities are shown in the format xdpi x ydpi, such as 203 x 200 dpi. This means that the printer prints 203 dots per horizontal inch, and 200 dots per vertical inch. In the case of multiple densities, a table with columns for XDPI, YDPI and XYDPI shows the dpi produced by each density. XYDPI is the number of dots per square inch; it is the result of multiplying the number of horizontal dots by the number of vertical dots.

There are also diagrams of the appropriate DIP switch or jumper settings for many of the printers. In the DIP switch diagrams, the correct position of the switch is indicated by the black box. For instance, in the diagram below, switches 1, 2, 3, 4, and 5 are off, and switches 6, 7, and 8 are on.



CalComp_ColorMaster

This driver can also be used with the ColorView-5912 printer.

- Thermal transfer black-and-white/color printer; prints text and graphics.
- One density is supported: 203 x 200 dpi. Selecting a density higher than 1 has no effect.
- This driver can be used with both the ColorMaster and ColorView-5912 printers. For the ColorMaster, set the Paper Size gadget to Narrow Tractor. For the ColorView-5912, set the Paper Size gadget to Wide Tractor.
- There are no DIP switches.

CalComp_ColorMaster2

- This driver is essentially the same as the CalComp_ColorMaster driver. However, it is approximately 2 times faster during color dumps although it requires a large amount of memory. For instance, a full 8 x 10 inch (1600 x 2000 dot) color dump requires approximately 1.2MB of memory. Typically, full-size color dumps are 1600 x 1149 dots and require approximately 700K.

Memory requirements for the ColorView-5912 can be as high as 2.5MB for a full 10 x 16 inch (2048 x 3200 dot) color dump. Typically, full-size color dumps are 2048 x 2155 dots and require 1.6MB.

Memory requirements for black-and-white or grey scale printing are approximately one-third of what is needed for a comparable color dump.

CBM_MPS1000

This driver can also be used with IBM5152 compatible printers. If you own a CBM MPS 1250, use the EpsonX printer driver.

- Dot matrix black-and-white printer; prints text and graphics.
- Multiple densities are supported:

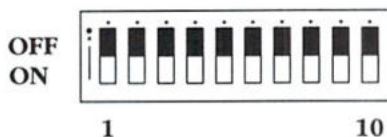
Density	XDPI	YDPI	XYDPI	Comments
1	120	72	8640	
2	120	144	17280	Performs two passes
3	240	72	17280	
4	120	216	25920	Performs three passes
5	240	144	34560	Performs two passes
6	240	216	51840	Performs three passes

- A density of 6 is the highest supported. Setting the Density gadget to 7 has the same result as setting it to 6.
- Switch settings:

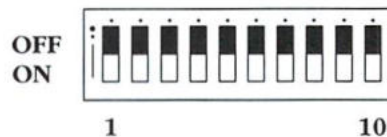
CBM MPS1000



Canon BJ-130 with Control Capsule 48/XL — IBM Proprinter® Compatible



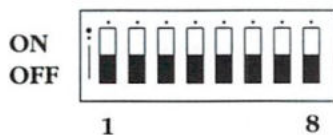
SW 1



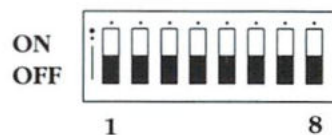
SW 2

Diablo_630

- Daisy-wheel printer; prints text only.
- Switch settings:



SW 1



SW 2

EpsonQ

This driver can be used with all the Epson® Q series compatible printers (LQ1500, LQ2500, etc.).

- 24-pin, dot-matrix black-and-white/color printer; prints text and graphics.
- Multiple densities are supported:

Density	XDPI	YDPI	XYDPI
1	90	180	16200
2	120	180	21600
3	180	180	32400
4	360	180	64800

- A density of 4 is the highest supported. Setting the Density gadget to 5, 6, or 7 has the same result as setting it to 4. When a density of 4 is selected, the printer cannot print two consecutive dots in a row. It is recommended that you only use this density for black-and-white printing.
- If the Paper Size gadget is set to Wide Tractor, the maximum print width for wide carriage printers is 13.6 inches.
- If the Paper Type gadget is set to Single, only 16 of the 24 pins are used. This option is useful for printers that have a weak power supply and cannot drive all 24 pins continuously. If you notice during a single pass of the print head that the top two-thirds of the graphics are darker than the bottom one-third, you should probably set the Paper Type to Single.
- If the Paper Type gadget is set to Fanfold, all 24 pins are used.

EpsonX

This driver can be used with the CBM MPS 1250 printer and all 8/9-pin Epson X series compatible printers (EX, FX, JX, LX, MX, RX, etc.).

If you are using an Epson X compatible printer and you notice that this driver does not work properly in graphics mode, try the EpsonXOld printer driver.

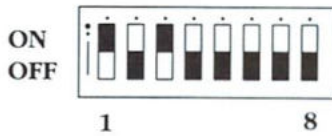
- Dot matrix, black-and-white and color printer; prints text and graphics.
- Multiple densities are supported:

Density	XDPI	YDPI	XYDPI	Comments
1	120	72	8640	
2	120	144	17280	Performs two passes
3	240	72	17280	
4	120	216	25920	Performs three passes
5	240	144	34560	Performs two passes
6	240	216	51840	Performs three passes

- A density of 6 is the highest supported. Setting the Density gadget to 7 has the same result as setting it to 6.
- When printing 240 xdpi (a density of 3, 5, or 6), the printer cannot print two consecutive dots in a row. It is recommended that you only use this density for black-and-white printing.
- If you're printing 72 ydpi (a density of 1 or 3), and you notice tiny, white horizontal stripes in your printout, try setting the Paper Type gadget to Single. In this mode, the line feed will be the number of vertical dots printed less one-third of a dot.

- If the Paper Size gadget is set to Wide Tractor, the maximum print width for wide carriage printers is 13.6 inches.
- Switch settings:

Commodore MPS 1250 Printer

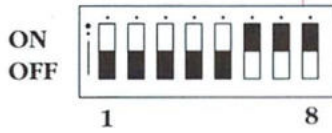


Serial/Parallel
Interface Pack

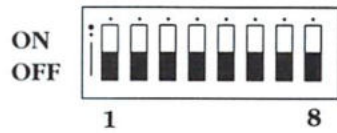


Printer Internal Switch

Epson EX-1000 Printer



SW 1

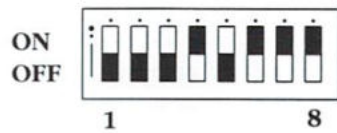


SW 2

Epson FX-80 Printer



SW 1



SW 2

EpsonXOld

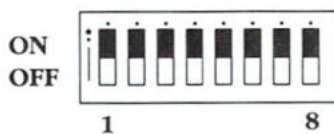
This driver is for older 8/9-pin Epson X compatible printers as well as the Star Micronics Gemini 10-X printer. If you are using an EpsonX compatible printer and you notice that the EpsonX driver does not work properly in graphics mode, try this driver.

- Dot matrix, black-and-white printer; prints text and graphics.
- Multiple densities are supported:

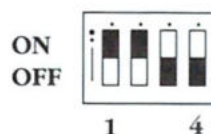
Density	XDPI	YDPI	XYDPI	Comments
1	60	72	4320	
2	120	72	8640	Double speed
3	120	72	8640	
4	240	72	17280	
5	120	72	8640	Use with Star printers
6	240	72	17280	Use with Star printers

- Setting the Density gadget to 7 has the same result as setting it to 4.
- When printing 240 xdpi (a density of 4 or 6), the printer cannot print two consecutive dots in a row. It is recommended that you only use this density for black-and-white printing.
- If the Paper Size gadget is set to Wide Tractor, the maximum print width for wide carriage printers is 13.6 inches.
- Switch settings:

Star Micronics Gemini 10-X



SW 1 (Internal)



SW 2 (External)

Howtek_Pixelmaster

- Plastic, ink-jet, black-and-white/color printer; prints text and graphics.
- Multiple densities are supported:

Density	XDPI	YDPI	XYDPI
1	80	80	6400
2	120	120	14400
3	160	160	25600
4	240	240	57600

- A density of 4 is the highest density supported. Setting the Density gadget to 5, 6, or 7 has the same result as setting it to 4.
- The maximum print area is 8.0 x 10.0 inches.
- There are no DIP switches.

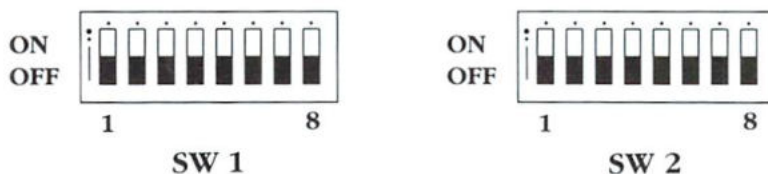
HP_DeskJet

- Ink-jet, black-and-white printer; prints text and graphics.
- Multiple densities are supported:

Density	XDPI	YDPI	XYDPI
1	75	75	5625
2	100	100	10000
3	150	150	22500
4	300	300	90000

- A density of 4 is the highest density supported. Setting the Density gadget to 5, 6, or 7 has the same result as setting it to 4.

- The maximum print area is 8.0 x 10.0 inches.
- Switch settings:



HP_LaserJet

This driver can be used with LaserJet Plus™ and LaserJet II compatible printers.

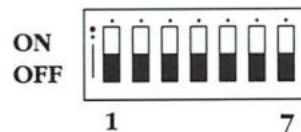
- Laser engine, black-and-white printer; prints text and graphics.
- Multiple densities are supported:

Density	XDPI	YDPI	XYDPI
1	75	75	5625
2	100	100	10000
3	150	150	22500
4	300	300	90000

- A density of 4 is the highest density supported. Setting the Density gadget to 5, 6, or 7 has the same result as setting it to 4.
- The maximum print area is 8.0 x 10.0 inches.
- There are no DIP switches.

HP_PaintJet

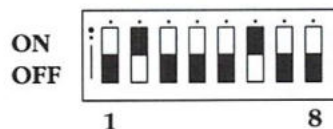
- Ink-jet, black-and-white/color printer; prints text and graphics.
- One density is supported: 180 x 180 dpi. Selecting a density higher than 1 has no effect.
- Switch settings:



HP_ThinkJet

- Ink-jet, black-and-white/color printer; prints text and graphics.
- Two densities are supported:

Density	XDPI	YDPI	XYDPI
1	96	96	9216
2	192	96	18432
- A density of 2 is the highest density supported. Setting the Density gadget to 3, 4, 5, 6, or 7 has the same result as setting it to 2.
- Switch settings:



ImagewriterII

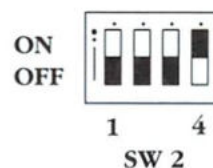
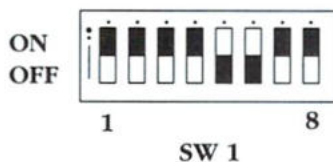
This driver can be used with Imagewriter™ printers.

- Dot matrix, black-and-white/color printer; prints text and graphics.
- Multiple densities are supported:

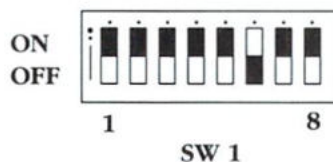
Density	XDPI	YDPI	XYDPI	Comments
1	80	72	5760	
2	120	72	8640	
3	144	72	10368	
4	160	72	11520	
5	120	144	17280	Performs two passes
6	144	144	20736	Performs two passes
7	160	144	23040	Performs two passes

- Switch settings:

Imagewriter



ImagewriterII



NEC_Pinwriter

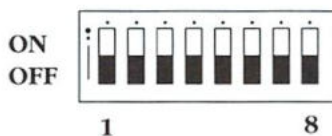
This driver can be used with all NEC® 24-wire Pinwriter® compatible printers (P5, P6, P7, P9, P2200, etc.).

- Dot matrix, black-and-white/color printer; prints text and graphics.
- Multiple densities are supported:

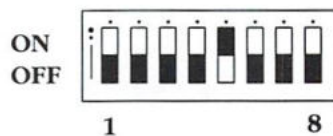
Density	XDPI	YDPI	XYDPI	Comments
1	90	180	16200	
2	120	180	21600	
3	180	180	32400	
4	120	360	43200	Performs two passes
5	180	360	64800	Performs two passes
6	360	180	64800	
7	360	360	129600	Performs two passes

- If the Paper Size gadget is set to Wide Tractor, the maximum print width for wide carriage printers is 13.6 inches.
- Switch settings:

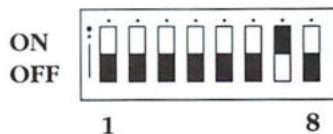
NEC Pinwriter P9XL



SW 1



SW 2



SW 3

Okidata_293I

This driver can be used with the Okidata® 292 or 293 printers with the IBM interface module.

- Dot matrix, black-and-white printer; prints text and graphics.
- Multiple densities are supported:

Density	XDPI	YDPI	XYDPI	Comments
1	120	144	17280	
2	240	144	34560	
3	120	288	34560	Performs two passes
4	240	288	69120	Performs two passes

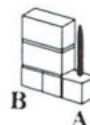
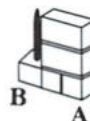
- A density of 4 is the highest density supported. Setting the Density gadget to 5, 6, or 7 has the same result as setting it to 4.
- If the Paper Type gadget is set to Single and you're printing 144 ydpi (a density of 1 or 2), line feeds are equal to the number of vertical dots printed less one-half of a dot. You may want to use this setting if you notice tiny white, horizontal stripes on your printout.
- If the Paper Size gadget is set to Wide Tractor, the maximum print width for wide carriage printers is 13.6 inches.
- Jumper settings:

ML-292/293 Personality Module

SP1



SP4



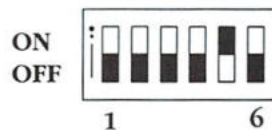
Okidata_92

- Dot matrix, black-and-white printer; prints text and graphics.
- One density is supported: 72 x 72 dpi. Selecting a density setting higher than 1 has no effect.
- Line feeds are always 7/72 of an inch.

Okimate_20

- Thermal transfer, black-and-white/color printer; prints text and graphics.
- One density is supported: 120 x 144 dpi. Selecting a density setting higher than 1 has no effect.
- Line feeds are equal to an even number of dots printed. For instance, if three dots were printed, four dots will be advanced.
- Switch settings:

Parallel Plug'n Print Kit



NOTE: Switch 5 on some models controls the white space between the lines of a graphic dump.

Serial Plug'n Print Kit



NOTE: The SW1 settings specify a baud rate of 9600, XON/XOFF handshaking, 8 bits, and no parity. On some models, switch 5 of SW2 controls the white space between the lines of a graphic dump.

Seiko_5300

- Thermal transfer, black-and-white/color printer; prints graphics only.
- Three densities are supported:

Density	XDPI	YDPI	XYDPI	Comments
1	152	152	23104	CH-5301 printer
2	203	203	41209	CH-5302 printer
3	240	240	57600	CH-5303 printer

- A density of 3 is the highest density supported. Setting the Density gadget to 4, 5, 6, or 7 has the same result as setting it to 3.
- There are no DIP switches.

Seiko_5300a

- This driver is essentially the same as the Seiko_5300 driver. However, it is approximately 2 times faster during color dumps although it requires a large amount of memory. For instance, a full 8 x 10 inch (1927 x 2173 dot) color dump requires approximately 1.5MB of memory. Typically, full-size color dumps are 1927 x 1248 dots and require approximately 875K.

Memory requirements for black-and-white or grey scale printing are approximately one-third of what is needed for a comparable color dump.

- There are no DIP switches.

Tektronix_4693D

- Thermal transfer, black-and-white/color printer; prints graphics only.
- One density is supported: 300 x 300 dpi. Selecting a density higher than 1 has no effect.
- Due to the way the printer images a picture, only the Aspect (Horizontal or Vertical) and Shade (Black-and-White, Color, or Grey Scale) settings of the PrinterGfx editor are recognized. As a result, only full-size pictures can be printed.
- For normal prints, use the printer's keymap to set the parameters specified below:

# of Window	Window	Parameter
3c	Printer Color Adjustment	Do Not Adjust
3d	Video Color Correction	Do Not Adjust
5	Background Color Exchange	Prints Colors as Received

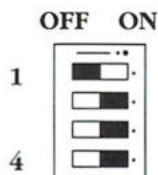
Texttronix_4696

- Ink-jet, black-and-white printer; prints text and graphics.
- Three densities are supported:

Density	XDPI	YDPI	XYDPI	Comments
1	121	120	14520	Outputs all colors in one pass
2	242	120	29040	Black-and-white; performs a double pass on black
3	242	120	29040	Color; performs a double pass on all colors

The densities 1 through 3 correspond to the printer's graphics printing modes 1 through 3, respectively.

- Selecting a density of 2 or 3 doesn't give you true 242 dpi resolution, since the printer only supports 121 dpi. Instead, it tells the printer to go into its double-pass mode. In this mode, it outputs a line of dots at 121 dpi, then it outputs the line again — shifted to the right by 1/242 of an inch. This produces more vibrant colors and give the illusion of a higher resolution. One drawback to this method is that large areas of solid colors, especially red, green and blue, tend to oversaturate the paper with ink.
- If the Paper Size gadget is set to Wide Tractor, the maximum print width for wide carriage printers is 9.0 inches.
- Switch settings:



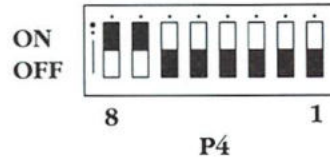
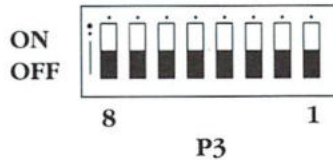
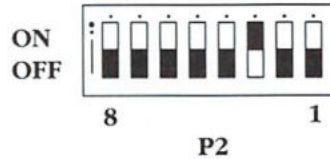
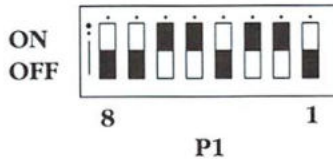
Toshiba_P351C

This driver can be used with all 24-pin Toshiba P351C compatible printers.

- Dot-matrix, black-and-white/color printer; prints text and graphics.
- Two densities are supported:

Density	XDPI	YDPI	XYDPI
1	180	180	32400
2	360	180	64800

- A density of 2 is the highest density supported. Setting the Density gadget to 3, 4, 5, 6, or 7 has the same result as setting it to 2.
- If the Paper Size gadget is set to Wide Tractor, the maximum print width for wide carriage printers is 13.5 inches.
- Switch settings:



Toshiba_P351SX

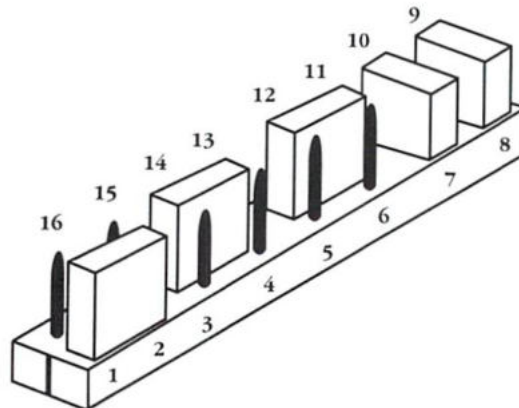
This driver can be used with all Toshiba P351SX compatible printers (321SL, 321SLC, 341SL).

- Dot matrix, black-and-white/color printer; prints text and graphics.
- Multiple densities are supported:

Density	XDPI	YDPI	XYDPI	Comments
1	180	180	32400	
2	360	180	64800	
3	180	360	64800	Performs two passes
4	360	360	129600	Performs two passes

- A density of 4 is the highest density supported. Setting the Density gadget to 5, 6, or 7 has the same result as setting it to 4.
- If the Paper Size gadget is set to Wide Tractor, the maximum print width for wide-carriage printers is 13.5 inches.
- Jumper settings:

If using the serial interface, set the jumpers in the following positions:

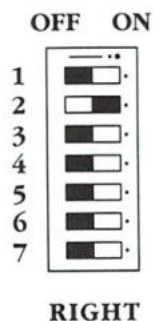
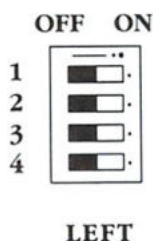


Xerox_4020

- Ink-jet, black-and-white/color printer; prints text and graphics.
- Two densities are supported:

Density	XDPI	YPDI	XYDPI
1	121	120	14520
2	242	120	29040

- Selecting a density of 2 does not give you true 242 dpi resolution. Instead, the printer outputs a line of dots at 121 dpi, then moves the paper up 1/240 of an inch and outputs the line again — shifted to the right by 1/240 of an inch. This produces more vibrant colors and gives the illusion of more resolution. One drawback to this method is that large areas of solid colors, especially red, green and blue, tend to oversaturate the paper with ink.
- Line feeds are always equal to 4 dots.
- If the Paper Size gadget is set to Wide Tractor, the maximum print width for wide roll paper is 9.0 inches.
- Switch settings:



Printer Escape Sequences

When typing an escape sequence from the keyboard, you press Esc. In BASIC, CHR\$(27) is used. In C, octal 033 can be used.

The Amiga print device (PRT:) accepts standard escape sequences that allow you to implement special printer features. For instance, you can use escape sequences to set margins, turn on styles (like boldface or italics), and specify spacing. (If the feature is not supported by your printer or printer driver, the escape sequence will be ignored.)

Escape sequences are typically used when you are printing to the printer device directly from the Shell or when you are inserting print commands into a program you are writing. These escape sequences are not necessary if you are using a word processor or desktop publishing programs, as you can specify the printing options through the program.

A typical escape sequence (to turn on boldfacing) is shown below:

```
Esc[1m
```

This means that you must press the following keys in sequence: Esc, [, 1, m. Escape sequences are case-sensitive. For instance, to enter the following escape sequence:

```
Esc[4W
```

you must press: Esc, [, 4, Shift, W.

If a number needs to be inserted into the escape sequence it is indicated by <n>. The n represents the number of your choice. Do not type the brackets; they simply indicate that a substitution must be made. For instance, the escape sequence to set the left and right margins is:

```
Esc[<n>;<n>s
```

If you wanted to specify a left margin of 5 and a right margin of 75, you would type:

```
Esc[5;75s
```

To send an escape sequence to the printer from the Shell:

1. Redirect the keyboard input to the printer by typing:

```
1> COPY * to PRT:
```

2. Wait until any disk access stops, then type an escape sequence, such as:

```
Esc[2"z
```

This sequence turns on the printer's NLQ (near letter quality) mode. You must press: Esc, [, 2, ", z.

To terminate the keyboard input, press Ctrl-\..

You can also create printer command files consisting of several escape sequences by redirecting the keyboard input to a file. For instance:

1. Redirect the keyboard input to a file:

```
1> COPY * TO RAM:EscapeFile
```

2. Type the escape sequences, such as:

Esc[2"z	(turns near letter quality on)
Esc[2w	(turns elite type on)
Esc[1m	(turns boldface on)
Ctrl-u	(terminates input)

3. To send these escape sequences to the printer, type:

```
1> COPY RAM:EscapeFile TO PRT:
```


Escape Sequences		
Feature	Escape Sequence	Name
Reset printer	Esc c	aRIS
Initialize printer	Esc #1	aRIN
Line feed	Esc D	aIND
Return line feed	Esc E	aNEL
Reverse line feed	Esc M	aRI
Normal character set	Esc [0m	aSGR0
Italics on	Esc [3m	aSGR3
Italics off	Esc [23m	aSGR23
Underline on	Esc [4m	aSGR4
Underline off	Esc [24m	aSGR24
Boldface on	Esc [1m	aSGR1
Boldface off	Esc [22m	aSGR22
Set foreground color	Esc [30m to Esc [39m	aSFC
Set background color	Esc [40m to Esc [49m	aSBC
Normal pitch	Esc [0w	aSHORP0
Elite pitch on	Esc [2w	aSHORP
Elite pitch off	Esc [1w	aSHORP1
Condensed fine pitch on	Esc [4w	aSHORP4
Condensed off	Esc [3w	aSHORP3
Enlarged pitch on	Esc [6w	aSHORP6
Enlarged pitch off	Esc [5w	aSHORP5
Shadow print on	Esc [6"z	aDEN6
Shadow print off	Esc [5"z	aDEN5
Doublestrike on	Esc [4"z	aDEN4
Doublestrike off	Esc [3"z	aDEN3
Near Letter Quality on	Esc [2"z	aDEN2
Near Letter Quality off	Esc [1"z	aDEN1
Superscript on	Esc [2v	aSUS2
Superscript off	Esc [1v	aSUS1
Subscript on	Esc [4v	aSUS4

Escape Sequences		
Feature	Escape Sequence	Name
Subscript off	Esc[3v	aSUS3
Normalize the line	Esc[0v	aSUS0
Partial line up	Esc l	aPLU
Partial line down	Esc K	aPLD
US character set	Esc[B	aFNT0
French character set	Esc[R	aFNT1
German character set	Esc[K	aFNT2
UK character set	Esc[A	aFNT3
Danish I character set	Esc[E	aFNT4
Swedish character set	Esc[H	aFNT5
Italian character set	Esc[Y	aFNT6
Spanish character set	Esc[Z	aFNT7
Japanese character set	Esc[J	aFNT8
Norwegian character set	Esc[6	aFNT9
Danish II character set	Esc[C	aFNT10
Proportional spacing on	Esc[2p	aPROP2
Proportional spacing off	Esc[lp	aPROP1
Proportional spacing clear	Esc[0p	aPROP0
Set proportional offset	Esc[<n> E	aTSS
Auto left justify	Esc[5 F	aJFY5
Auto right justify	Esc[7 F	aJFY7
Auto full justify	Esc[6 F	aJFY6
Auto justify off	Esc[0 F	aJFY0
Letter space (justify)	Esc[3 F	aJFY3
Word fill (auto center)	Esc[1 F	aJFY1
1/8" line spacing (8 lpi)	Esc[0z	aVERP0
1/6" line spacing (6 lpi)	Esc[1z	aVERP1
Set form length to <n>	Esc[<n>t	aSLPP
Perf skip <n> (n > 0)	Esc[<n>q	aPERF
Perf skip off	Esc[0q	aPERF0
Left margin set	Esc#9	aLMS
Right margin set	Esc#0	aRMS

Escape Sequences		
Feature	Escape Sequence	Name
Top margin set	Esc#8	aTMS
Bottom margin set	Esc#2	aBMS
Top and bottom margins	Esc[<n>;<n>r	aSTBM
Left and right margins	Esc[<n>;<n>s	aSLRM
Clear margins	Esc#3	aCAM
Set horizontal tab	EscH	aHTS
Set vertical tabs	EscJ	aVTS
Clear horizontal tab	Esc[0g	aTBC0
Clear all horizontal tabs	Esc[3g	aTBC3
Clear vertical tab	Esc[l g	aTBC1
Clear all vertical tabs	Esc[4g	aTBC4
Clear all horizontal and vertical tabs	Esc#4	aTBCALL
Set default tabs	Esc#5	aTBSALL
Extended commands	Esc[<n>"<x>	aESTEND

An extended command allows you to specify a *printer specific* command. This is a command that is recognized by your printer, not by the Amiga, such as a command to use a particular font. In this case, <n> represents the number of bytes in the command, and <x> represents the actual command. For instance, if your printer recognizes Esc-k-1 as the command to use a sans serif font, you would type:

```
Esc[3"Esc k1
```

If you are entering extended commands within a program you are writing, make sure that the program can only be used with one specific printer. If you enter extended commands for an Epson printer, then someone tries to use the program with an HP LaserJet, the command may not work.

Appendix C

Backing Up Your Hard Disk with BRU

BRU is a backup and restore utility included on the Extras disk. There is no icon for BRU; it is accessed only through the Shell. BRU allows you to back up (archive) your hard disk by copying information stored on the hard disk to floppy disks or magnetic tape (if you have a magnetic tape drive). If backed-up files are ever lost because of hardware or software failures or human errors, you can easily restore recent versions of your programs and files to your system.



BRU offers the Shell user many advanced options not accessible from the HDBackup program described in Chapter 6 of this manual.

This appendix tells you how to customize BRU's defaults through Brutab, a data file that provides BRU with information about the archive devices; provides a basic tutorial to walk you through BRU's most common command lines; gives individual explanations of all BRU commands; and explains BRU's error messages.

NOTE: BRU was designed for use on both Amiga and UNIX operating systems. Because of this, the format of BRU command lines differs from the standard AmigaDOS format. Also, some commands may be specific to UNIX operations and can be used with archives imported to AmigaDOS from UNIX. These commands are listed separately in the "BRU with UNIX" section beginning on page C-36.

Brutab

Before you begin to use BRU, you should understand its default structure. Brutab is the ASCII file that provides BRU with basic information such as the device you are archiving to (a floppy drive, for example). Brutab also includes the size characteristics of each device and error recovery values that your own system uses.

Brutab is located in the S: directory and looks something like this:

```
df0:
size = 880K seek = 512 bufsize = 22K noreopen qfwrite \
prerr = 5 pwerr = 5 zrerr = 5 zwerr = 5 frerr = 5 fwerr = 5 wperr = 30 \
rawfloppy device = trackdisk.device unit = 0
df1:
size = 880K seek = 512 bufsize = 22K noreopen qfwrite \
prerr = 5 pwerr = 5 zrerr = 5 zwerr = 5 frerr = 5 fwerr = 5 wperr = 30
rawfloppy device = trackdisk.device unit = 1
df2:
size = 880K seek = 512 bufsize = 22K noreopen qfwrite \
prerr = 5 pwerr = 5 zrerr = 5 zwerr = 5 frerr = 5 fwerr = 5 wperr = 30
rawfloppy device = trackdisk.device unit = 2
df3:
size = 880K seek = 512 bufsize = 22K noreopen qfwrite \
prerr = 5 pwerr = 5 zrerr = 5 zwerr = 5 frerr = 5 fwerr = 5 wperr = 30
rawfloppy device = trackdisk.device unit = 3
tape:
size = 0 seek = 0 bufsize = 200K noreopen rewind \
prerr = 5 pwerr = 5 zrerr = 5 zwerr = 5 frerr = 5 fwerr = 5 wperr = 30 \
advance tape rawtape format device = scsi.device unit = 3
ntape:
size = 0 seek = 0 bufsize = 200K noreopen norewind \
prerr = 5 pwerr = 5 zrerr = 5 zwerr = 5 frerr = 5 fwerr = 5 wperr = 30 \
advance tape rawtape format device = scsi.device unit = 3
—
size = 0 seek = 0 \
prerr = 0 pwerr = 0 zrerr = 0 zwerr = 0 frerr = 0 fwerr = 0 wperr = 0
```

The Components of the Brutab File

Normally the values in Brutab will not need to be changed, however you may choose to customize Brutab to suit your individual needs. You will need to use a text editor to make changes to the file. Only experienced users should attempt to customize Brutab.

Entries in Brutab consist of multiple fields separated by white space (spaces or tabs). A \ (backslash) character at the end of a line indicates a continuation of the entry to the next line. Each entry consists of a device name, followed by one or more capabilities fields (explained below). All tabs and blanks between fields are ignored.

Brutab lists the possible devices to which you could archive. The first four entries (df0: through df3:) are floppy disk drives; the fifth (tape:) is magnetic tape; and the sixth (ntape:) is non-rewinding magnetic tape. The last entry signifies the end of the Brutab file.

BRU uses the first device entry in Brutab as its default setting—it will then archive to that device. If the default setting is correct (in the case of the example Brutab on page C-2, if you are planning to archive to floppy disks in drive DF0:) you may never need to change your Brutab file.

If you will not be archiving with disks in drive DF0:, you will need to move the correct listing first, or specify a drive other than the default by using the -f <path> option described later in this appendix.

After the device name field, each subsequent field defines either a numeric or a **boolean** (on/off; true/false) characteristic of the device, in this form:

capability = value (such as "size = 640K") or
boolean flag (such as "tape")

There must be no white space between the capability name and the value.

Numeric values may be given in absolute form or appended with a scale factor of:

b or B	Blocks (512 bytes)
k or K	Kilobytes (1024 bytes)
m or M	Megabytes (1024*1024 bytes)

The following is a list of the Brutab fields:

size	Media size of the archive device in bytes, if known. Zero if unknown or variable sized media (such as a tape drive which can take various sized tapes). If a size is given, you must not attempt to use a media with a real capacity less than this size.
seek	Minimum seek resolution. All seeks performed on the device will be an integral multiple of this value. Zero if no seeks allowed.
bufsize	Default I/O buffer size for this device. If omitted, the default is 22K.

The following values represent **error numbers**, variables used by BRU to store error results. For more information, see page 8-131.

prerr	Value left in Result2 for partial reads. A partial read is one that successfully reads more than zero bytes but less than the requested number of bytes.
pwerr	Value left in Result2 for partial writes. A partial write is one that successfully writes more than zero bytes but less than the requested number of bytes.

zrerr	Value left in Result2 for zero length reads. A zero length read is one that reads zero bytes.
zwerr	Value left in Result2 for zero length writes. A zero length write is one that writes zero bytes.
frerr	Value left in Result2 after an attempt to read from unformatted media.
wperr	Value left in Result2 after an attempt to write to a write protected media.

The boolean fields are:

reopen	If specified, BRU will close and reopen archive upon media switch. In some instances, if the archive is not closed and reopened, the system will refuse to access certain devices after the media has been removed.
noreopen	If specified, BRU will not close and reopen archive upon media switch. It will leave the archive device open across media switches.
tape	Indicates that the archive device is a tape drive. This includes 9-track tapes and streaming tape drives.
rawtape	Indicates that the archive device is a "raw" tape drive and that buffering is not to be used when sending data to and from the device.

norewind	If specified, the device will not automatically rewind back to the start of the media (applies to most tape drives) upon closing. <i>The size parameter should be set as zero to use this feature.</i>
advance	If specified, BRU reads and writes advance media even when errors occur. True for many tape drives.
qfwrite	If specified, BRU will request confirmation to proceed for the first write to the first medium in this device. This flag should be used to protect against accidentally overwriting a disk that may have been left in a floppy drive, for example.
eject	If specified, BRU will eject media after use (if the device supports auto-ejection).
format	If specified, BRU will automatically format media if the first write to the media fails and BRU knows how to format media for this type of device.

Setting Environment Variables for BRU

Instead of customizing Brutab each time you archive in a different manner, you may wish to have many different versions of Brutab. Normally, BRU looks in the S: directory to find Brutab, but you can override this by setting your environment variable Brutab to indicate the specific version of Brutab that you wish to use. To do this, you will need to use the SETENV command.

In this example, Brutab.2 is your second Brutab file also located in the S directory. To instruct BRU to use this version, type:

```
SETENV BRUTAB = sys:s/brutab.2
```

To instruct BRU to revert to your original Brutab file, type:

```
UNSETENV BRUTAB
```

For more information on the SETENV command, see page 8-114.

BRU Command Lines

This section explains the format of BRU commands.

Before you can use BRU through the Shell, you must set at least a 25000 byte stack. If you are opening a Shell without this stack size, you must type:

```
1> STACK 25000
```

You must do this every time you open a Shell to use BRU.



Next, a command line beginning with the word BRU is entered, followed by one character arguments preceded by a hyphen, such as -c, -v, and -f. These arguments tell BRU which functions you would like it to perform. (Complete descriptions of these arguments are found in the "BRU Argument Reference Section" beginning on page C-19.)

A BRU command line looks like this:

BRU <modes> [control options] [selection options] [files]

where:

BRU	The command that precedes any BRU arguments you enter; it invokes the BRU program.
<modes>	Arguments that tell BRU to perform its basic functions. The modes are BRU's fundamental actions, such as creating an archive, estimating the size of an archive, and extracting files. <i>In any BRU command, you must specify at least one mode.</i> You may choose a number of modes and list them all together.
[control options]	Optional arguments that tell BRU <i>how</i> something is done. If no control options are specified in the command line, BRU uses the system defaults, such as those set in Brutab. Control options include selecting a new device on which to archive, changing the size of the archive, and so on. You may choose a number of control options and list them all together.
[selection options]	Optional arguments that tell BRU <i>which</i> files to select. An example of a selection option is telling BRU to save only files which have been created or changed after a particular date. You may choose a number of selection options and list them all together.

[files]

The optional file list that comes after all of the modes and options that you have already specified. The file list tells BRU exactly which partitions, directories and/or files you want to either save or restore to your computer. More than one file may be listed, with a space to separate each one. *If no files are specified, BRU archives from the current directory.* BRU scans its command line and interprets any entries following modes, control options, and selection options as file names.

BRU is case sensitive—you must properly use upper and lowercase in arguments and file names.

Beginning users may feel comfortable archiving and restoring with BRU modes. It is recommended that only advanced users attempt incorporating control and selection options in BRU commands.



Interrupting BRU While it is Operating

If you need to stop BRU in mid-operation, press Ctrl-C.

Using BRU—A Tutorial

This section will walk you through commonly used BRU command lines. There are many more arguments to choose from than demonstrated here. All of the BRU arguments are listed and explained in the following section, the "BRU Argument Reference Section," beginning on page C-19.

Estimating the Size and Creating a Backup of the Current Directory

Normally, the first step in creating an archive will be deciding what files and directories you wish to back up. Next, you will want to estimate the size of the archive so you can have the proper number of disks on hand before you begin.

NOTE: Disks do not have to be formatted before using them with BRU.

The mode for estimating the size of an archive is `-e`.

Unless partitions, directories and/or files are specified in the file list on the command line, BRU performs all actions on the current directory. In this first example, you will estimate the size of an archive of the current directory (normally `SYS:`).

1. Open a Shell and type:

```
1> STACK 25000  
1> BRU -e
```

BRU will respond with an estimate similar to this:

```
bru: 5 volume(s), 396 files, 1808 archive blocks, 3616 Kbytes
```

BRU has told you the total size of the archive, and that you will need 5 volumes (disks) to hold this archive.

Once you know how big your archive will be, you may be ready to create it. The argument for creating an archive is `-c`. The following command instructs BRU to create an archive of the current directory:

1. Type:

```
1> BRU -c
```

(Note that since again no partitions, directories or files were listed at the end of the command line, BRU will perform all actions on the current directory.)

BRU will respond with something like this:

```
bru: warning - all data currently on "df0:" will be destroyed
bru: c => continue q => quit r => reload s => shell [default: r] >>
```

At this point, BRU waits for confirmation to proceed. You could type:

- | | |
|---|--|
| c | To proceed with the archive. |
| q | To quit, and return to the Shell prompt. |
| r | To pause to reload a disk. |
| s | To open a new Shell if you need to get additional information. |

2. If you wish to perform an archive of this type, press C and Return. (To quit, press Q and Return, and you will be returned to the Shell prompt.)

If you press C to continue, BRU will prompt you to insert volume one and enter the device to which you will archive. (If you press return, BRU will use the specified default.)

NOTE: Before archiving, you should always label your disks with three important pieces of information:

- The volume number of the disk.
- The partition you have backed up.
- The date.

3. *Label the first disk, insert it and press Return.*

BRU will continue to prompt you to reload volumes until the archive is complete. You will then be returned to the Shell prompt.

Backing up Your Entire Hard Disk

You may wish to back up your whole hard disk, rather than just the current directory. To do this, you will need to list the names of all of your hard drive's partitions to the file list at the end of the command line (with a space to separate each one).

For example, let's say you have a hard disk with two partitions: System: and Work:. You want to create an archive which spans your whole hard disk. You may first want to see how many disks this will require. To do this:

1. *At the Shell prompt, type:*

```
1> BRU -e System: Work:
```

BRU will respond with a size estimate of both partitions. Next, you will create the archive:

2. *Type:*

```
1> BRU -c System: Work:
```

BRU will warn you that it will erase all information on your default archive device (normally DF0:) and will prompt you to load disks as in the previous example. Remember to label your disks appropriately, and follow the prompts until your archive is complete.

Backing up Only a Few Files or Directories

With BRU, you don't always have to back up entire partitions. Let's say you've been working on a database every day for two months and don't want to risk losing it. Even though you back up your entire hard disk once a week, you want to back up this database separately every day. All you need to do is add that filename to the end of the BRU command line, and BRU will perform all actions on that file alone.

For example, let's say the database is called ClientBase, and it is located in the Work: partition. To create an archive of it, you would type:

```
1> BRU -c Work:ClientBase
```

Or you could CD to the Work: partition and type:

```
1> BRU -c ClientBase
```

In another example, let's say you have a directory in Work: called Clients which, among other things, contains a profile for each of your many clients. Each of those filenames ends in .prof (such as Smith.prof, Adams.prof, and so on). You've just hired a new associate and want an easy way to pass your client profiles to her. You could use a wildcard backup to back up only those files that end with the pattern .prof.

BRU uses the * character as a wildcard. To backup only those files that end in .prof you would type:

```
1> BRU -c Work:Clients/*.prof
```


Combining Modes

As mentioned in the “BRU Command Lines” section, BRU modes can be strung together on one command line. This means in one step you can instruct BRU to perform a series of actions.

In this example you will combine three arguments. The `-i` mode instructs BRU to inspect an archive for accuracy after it has been completed. Since `-e`, `-c`, and `-i` are all modes, you can string them together. In the following step, one command line you will tell BRU to estimate the size of an archive, create the archive, and then inspect the archive:

```
1> BRU -eci
```

NOTE: Modes do not have to be entered in any particular order; BRU executes them in order of priority. The order in which BRU executes modes is:

```
-ecitxdgh
```

For example, Estimate will be done before Create, even if the mode is specified as `-ce`.

Adding Control and Selection Options

For more control over archiving, control and selection options can be added to the BRU command line. These arguments may also be strung together, with a single space to separate modes, control options and selection options. (For a refresher on the format of BRU command lines, see the “BRU Command Lines” on page C-7.) This section will provide a few commonly used examples of control and selection options.

By adding the `-v` control option to any BRU command, BRU will become **verbose** (talkative) and display each action it takes as it occurs.

The following step illustrates combining the `-e` mode with the `-v` control option, estimating the size of the `Work:` partition:

1. Type:

```
1> BRU -e -v Work:
```

BRU will scroll through each file of the selected directory as it tallies up a final estimate. A section of this estimate may look like this:

```
e    4k of      6k [1] Work:HandShake.info
e    2k of      8k [1] Work:Trashcan
e    4k of     12k [1] Work:Trashcan.info
e    4k of     16k [1] Work:Disk.info
e    4k of     20k [1] Work:.info
e    2k of     22k [1] Work:ClientBase
e    6k of     28k [1] Work:ClientBase/Williams.prof
```

```
bru: 1 volume(s), 7 files, 270 archive blocks, 540 Kbytes
```

From left to right, the line lists the mode being used (`e` in this case), the size of the file, the total size of the archive (up to that point), the volume number in the archive, and the name of the file. BRU displays the total size of the archive at the end.

You may add the `-v` control option to any BRU mode.

Another commonly used control option labels your archive with any name you'd like (up to 63 characters). The argument for this control option is `-L <str>`, where `<str>` is the text string to label your archive. This is useful if you create a separate archive for each partition on your hard drive, for example. That way, the name of the archive will be contained on the archive disk.

The following BRU command line creates an archive of the Work: partition, labels it WorkArchive and shows you each step you are taking along the way (combining two control option arguments—`-v` and `-L <str>`):

```
BRU -c -vL WorkArchive Work:
```

The label WorkArchive is now present on the archive header on the disks. The `-g` mode allows you to see the archive header. To see the archive header:

1. Load the first volume of the archive.

2. Type:

```
1> BRU -g
```

BRU will respond with the archive header, which will look something like this:

```
1> BRU -g
label:           WorkArchive
created:         Mon Sep 24 12:04:19 1990
device:         df0:
user:           root
group:          root
system:         unknown unknown AmigaDos vm unknown
bru:            "Amiga Release 1.1"
release:        11.21
variant:        1
bufsize:        22528
msize:          901120
```

As mentioned earlier, selection options tell BRU *which* files to select from the current directory or from a specified directory. The argument `-n <date>` is the most commonly used selection option. It tells BRU to only archive files that were modified or created after the date and time you provide in the argument.

Let's say you archive your Work: directory every Friday at 5:00, and last Friday's date was September 28, 1990. The following example BRU command line will combine BRU modes, control options and selection options. You could tell BRU to

estimate the size of the Work: partition (-e), create an archive (-c), label it WorkArchive (-L <str>), and only archive files that changed on Work: since last Friday (-n <date>) by typing:

```
BRU -ec -L WorkArchive -n 9-28-90,17:00:00 Work:
```

NOTE: There are four formats you can use to specify the date with the -n <date> argument. For more information, see its detailed explanation on page C-34.

Restoring Files to Your Hard Disk

When you need to restore files to your hard disk, you must first decide where you want them. Where the files are placed in your system is determined by the way you created your archive—specifically, whether you stated a **relative path** or an **absolute path** in the file list.

A relative path does not include the name of the partition. *If you created an archive of the current directory, the archive was created with the relative path.* When you restore files created this way, you must change to the directory where you want those files to go before restoring them.

An absolute path is the full path of a file or directory, and includes the partition name. If you specified an absolute path when you created an archive (such as adding Work: to the end of the BRU command line), all filenames on the archive are prefaced with Work: and will always be restored to that path.

For example, assume you are creating an archive of all files in the directory DH0:devs/keymaps. You want to be able to put the files into a different directory when you restore. When you create that archive, you would first CD to the DH0: directory, and place devs/keymaps/* in the BRU command line's file list. This makes devs/keymaps/* the relative path (with *

representing all files within that directory]. Before you restore them, you would have to CD to the directory where you want the files.

Now let's say you are archiving the same directory, but you *always* want files to be restored to their original locations in DH0:, regardless of what directory you are in when you restore. In this case, you can create the archive from any directory, but must specify the absolute pathname in the file list (which is DH0:devs/keymaps/*). In this case, when you restore, those files will always be restored to drive DH0:.

The advantage of specifying a relative path is that a BRU disk can be used to restore to a different directory than it was created from or to a completely different system. It is the most versatile method because files can be restored anywhere on any system. If you want files restored to their original locations, you simply change to that directory.

NOTE: When you restore files, if the specified file in the archive is found to be older (an earlier date) than the current file on disk, BRU will not restore it to your hard disk. This default behavior avoids accidentally overwriting new files with old files from an archive. The behavior can be overridden with the -u option described in the BRU Argument Reference section.

Before restoring files, you may want to check the table of contents of an archive to see what it contains (also to see if the files were archived with a relative or absolute pathname). The mode that tells BRU to list a table of contents is -t. To do this:

1. *Load the first volume of the archive.*
2. *Open a Shell and type:*

```
1> STACK 25000  
1> BRU -t
```

BRU will list the table of contents of that disk (or tape). BRU will then prompt you to reload if you have a multi-volume archive.

Sometimes you will need to restore the entire archive to your hard disk. For instance, you may have lost everything in a partition by formatting it accidentally. To restore the entire archive:

1. *Change to the partition and/or directory where you want the files.*

2. *Type:*

```
1> BRU -x
```

Often, however, you will have to restore only one or two files from the archive. To do this include the appropriate files in the file list at the end of the BRU command line. You may list as many files as you wish, but you must put in a space between each one, such as:

```
1> BRU -x clients/smith.prof supplies/office supplies/retail
```

BRU Argument Reference Section

This section provides individual explanations of all BRU commands.

Modes

Creating an Archive

-c

Regardless of all the options you have available with BRU, a command line as simple as this can be used to save everything BRU can find in the current directory and within all subdirectories:

BRU -c

If you wish to create an archive of a *different* directory, either first change to that directory with the CD command or specify that directory in the file list at the end of the BRU command line.

Differences in size

-d

With the -d mode, BRU detects and reports differences between archived files and the files of the same name on your current system. This is useful to verify an archive immediately after it has been created. After an archive has been created, you can use the -d mode to monitor which files have changed since your last archive.

You should have the first volume (floppy disk or tape) in a drive before you enter this option. When BRU is finished reading the archive, it will print the list of files that differ between the archive and your present directory, along with a slight explanation of how the files are different. If no files are printed, there are no differences.

You can use this option at the creation of the archive, such as:

```
BRU -cd [options] [files]
```

and BRU will check through the archive after it is created. Or you can use it alone at a later date, such as:

```
BRU -d
```

to check which archived files might have changed since this archive was created.

The -d mode can be used in one of four forms (-d, -dd, -ddd or -dddd). The more -d's that are specified, the more detailed the error message output. With a single -d specified, BRU includes messages such as:

```
bru: "PRINT": no file: No such file or directory
```

```
bru: "PRINT/pscript.ps": no file: No such file or directory
```

```
bru: "PRINT/{sys}.prt": no file: No such file or directory
```

Estimating the Size

-e

Normally the first argument used, the -e mode tells BRU to make a quick scan across the files you have specified to archive (the current directory if there are no files listed at the end of the command line). Based on the type and size of the archive media specified in Brutab (floppy disks, for example), BRU will tell you the size of the archive and how many disks (or tapes) will be needed to create this archive.

To estimate the size of an archive in the current directory, type:

```
BRU -e
```

BRU will respond with an estimate similar to this:

```
bru: 5 volume(s), 396 files, 1808 archive blocks, 3616 Kbytes
```

In this case, you must have five disks to perform your archive.

NOTE: The estimate mode cannot be used if file compression is also selected. BRU would actually have to compress each file just to find out how much space it would take up in the archive. If you must have an estimate on an archive that uses file compression, you can create an archive and redirect it to the NIL: device by using:

```
BRU -cv -Z -f NIL: [files]
```

This command will write a compressed archive to the NIL: device and the verbosity output would tell you how many volumes would be needed to actually save the archive. Information on file compression (-Z), verbosity (-v) and the -f <path> option is provided later in this appendix.

Give Information on Archive Header

-g

The -g mode does not perform any archive actions. BRU simply lists archive header information on the current archive. To see the archive header, have the first volume of the archive loaded and type:

```
BRU -g
```

BRU will provide the archive header, such as:

label:	MyArchive
created:	Mon Jan 12 11:22:41 1987
device:	df0:
user:	root
group:	root
system:	unknown unknown AmigaDos vm T unknown
bru:	"Amiga Release 1.1"
release:	11.20
variant:	1
bufsize:	22528
msize:	901120

Print the BRU Help Screen

-h

Typing the argument:

```
BRU -h
```

will display a quick reference page describing the various BRU command line options.

Inspecting an Archive

-i

Using the `-i` mode tells BRU to read the entire archive to make sure it is intact. You may want to include this mode when you create your archive to make sure that it can be read back later. A defect in your archive could result in data loss when you attempt to restore files from your archive.

To use this option, you may include `-i` in the command line when you create an archive, such as:

```
BRU -ci [options] [files]
```

Note that you can tell BRU to inspect an archive at any time, not only immediately after creating it. You should load the first volume of your archive before you specify the `-i` command, as follows:

```
BRU -i
```

BRU will provide you with a list of any files which contain errors and short explanation of which errors have occurred.

Listing the Table of Contents

-t

The **-t** mode gives a listing of the table of contents of an archive. This option interacts with the **-v** control option (described later in this appendix) by providing you with more detailed information if **-v** or **-vv** are specified with the **-t** mode, such as:

```
BRU -t -v
```

or

```
BRU -t -vv
```

For example, when running the **-t** mode without any **-v** option, the table of contents is simply the list of archived files:

```
1> BRU -t  
  
PRINT/pscript.ps  
PRINT/{sys}.prt  
PRINT/Print  
PRINT/Print.info  
PRINT/Readme.Print  
PRINT/Readme.Print.info  
PRINT/PrintDef
```

Adding a single **-v** option produces a more specific table of contents (note the file name on the right):

```
1> BRU -t -v  
  
-rwxrwxrwx 1 root root 6915 Jan 11 1987 PRINT/pscript.ps  
-rwxrwxrwx 1 root root 1598 Jan 11 1987 PRINT/{sys}.prt  
-rwxrwxrwx 1 root root 37144 Jan 11 1987 PRINT/Print  
-rwxrwxrwx 1 root root 478 Jan 11 1987 PRINT/Print.info
```

The **-vv** option, when used with **-t**, provides information about hard or symbolic links, as well as file sizes. (*Note:* The Amiga does not currently support hard or symbolic links. This pertains to archives imported from UNIX operating systems.) You can add up to four **-v** options, producing an even more detailed output.

The first volume of your archive should be loaded before specifying these commands.

Extraction

-X

Extraction is the process of *restoring* files from an archive to your system. A command as simple as this restores an entire archive to your system:

```
1> BRU -x
```

Where the files get placed in your system is determined by the method you used to create your archive. If you created an archive of the current directory or if you specified a relative path on the command line, before you restore files you must change to the directory where you want those files to go. If you specify an absolute path when you create an archive, files are always restored to that path. For more information and examples see "Restoring Files to Your Hard Disk" on page C-17.

You may also choose to restore specific files from an archive. There are several ways you may choose to specify which files should be extracted. Among them are:

- using a wildcard file name expansion
- specifying a file date
- using flags to specify how to extract files

These commands are selection options, explained later in this appendix. In extraction, they are used in the same way they are used when creating an archive.

Control Options

Amiga Specific Flags

-A <flags>

This control option tells BRU how to handle pre-set archive bits as well as other Amiga-specific options. Current flags are:

- c Clear file archive bit after processing.
- f During filter mode reroute interactions to message port.
- i Ignore file archive bit for selecting files.
- r Reject files that have archived bit set.
- s Set file archive bit after processing.

Set Archive Buffer Size

-b <n>

To speed up the process of archiving, this option allows you to set the archive buffer size to <n> bytes. The minimum is the size of an archive block (2048 bytes) and the maximum is determined by available memory and I/O device limitations. If the byte size is not an even multiple of 2048 bytes, it will be rounded up. Normally this option is only required with the -c mode since BRU writes this information in the archive header block. If specified, this byte size overrides any existing default built in or read from the archive header.

Telling BRU to Run Without User Intervention

-B

Using the -B option forces BRU to run "in the background," expecting no user interaction. Perhaps before going home for the night, you will insert a backup disk (or tape) in one or more drives to do a nightly backup. Normally, if there is a user

present, certain errors might be recoverable (such as an unexpected disk full condition where BRU could have asked you to insert another disk). The -B option says there is no one here to ask for help, so BRU should terminate with an appropriate error code instead of asking for, then waiting for, user intervention.

If you or someone else is available during a backup, you may decide not to use this option to guarantee that the backup actually takes place.

Use Path as the Archive File -f <path>

The -f option is used to tell BRU to use multiple files or paths (which could be devices, for example) to store the archives. This feature is known as device cycling. If multiple -f options are given, each device is added to a list to cycle through each time a volume change is required. When the end of the list is reached, BRU automatically cycles back to the first device and waits for confirmation to continue the cycle again. Any input other than a carriage return will cause BRU to use the newly entered path and to abort the cycling for the remainder of the current run.

For example, if you want BRU to switch from drive DF0: to drive DF1: when creating an archive, the command reads as follows:

```
BRU -c -f df0: -f df1: [options] [files]
```

BRU will alternate between both floppy drives to back up the archive file, starting with DF0:.

Fast Mode

-F

In fast mode, check sum computations and comparisons are disabled. This mode is useful when the output of one backup is piped to the input of another backup, or when the data integrity of the archive transmission medium is essentially perfect. Archives recorded with fast mode enabled must also be read with fast mode. Be aware that some of the automatic features of BRU, such as automatic byte swapping, are not functional in fast mode.

Interaction Option

-I <option>

Using the interaction option allows you to run BRU from programs that automatically run tasks at specified times and frequencies. If no interaction with the user is required, the effect is no different than running BRU as a background task. However, when interaction is necessary there are two options: to either terminate the program or to wait for a response to a prompt. The -B option provides for simple termination while the -I option provides for communication with a user.

BRU recognizes the following parameters for the -I option:

l,pathname	write verbosity info to pathname
q,pipe (or file)	write interaction queries to a named pipe or file
r,pipe (or file)	read interaction queries from named pipe or file

You can use the device cycling feature (**-f <path>**) to specify that more than one archive device contains disks that are ready for an unattended backup. For example, you may load a set of disks into several drives, and schedule AREXX to begin a daily backup at midnight. BRU will do as much work as it can without interaction, and then wait for you to provide any additional information in the morning.

Labeling an Archive

-L <str>

You can label your archive by using the -L control option. This information appears in the archive header which can be viewed by using BRU's -g mode.

The following command line causes the label Test Backup to be attached to the archive:

```
BRU -c -L "Test Backup" [options] [files]
```

Note that the label is enclosed in a pair of double quotes. You must use a quote around labels contain a space in the name. This ensures that the label will be seen as a combination of two or more words. If you don't use the quotes, BRU would see the word Test as the label and treat the word Backup as a filename.

Labels can be a maximum of 63 characters.

Use nbits for Compression

-N <nbits>

This is a file compression option used in conjunction with the -Z command (discussed later in this appendix). *N* is the number of bits used for compression. The default is 12 bits, which is also the minimum allowable. The maximum allowable value is 16 bits. Archives created with more than 12 bits of compression may be unreadable on smaller systems due to memory or processor work length constraints.

Amiga users with only 512K of memory should not set compression over 12 bits.

Pass Over Archive Files by Reading

-p

Normally BRU will use random access capabilities if available. The -p option forces BRU to pass over files in an archive by reading rather than seeking.



Pathname Handling and Expansion -P <flags>

The -P option with a flag provides special options for pathname handling and expansions. Flags are:

- | | |
|---|---|
| e | Turn off expansion of directories |
| E | Turn on expansion of directories |
| f | Turn off filter mode (build internal file tree) |
| F | Turn on filter mode (do not build internal tree) |
| p | Turn off auto archiving of parent directory nodes |
| P | Turn on auto archiving of parent directory nodes |

Exclude Remotely Mounted Files -R

If your computer is connected to a network (such as Ethernet) with mounted file systems from remote computers, your archive could end up being very large. You may only want to back up the system on a single machine and not reach out across the network to other machines. The -R option tells BRU to exclude remote files from the archive.

If the system does not support remote filesystems, this option is ignored.

Specify Size of Archive Media -s <n>

Although BRU reads the size of the archive media from Brutab, this can be overridden with the -s <n> option, where <n> is the new media size specified in bytes. For example, you may be archiving to tape and have a few different sizes of tapes you use for backups. Each time you run BRU you can specify which kind of tape you'll be using. When the -s option is specified, it overrides any other default value, even one that is read from the tape header during a read or scan of the archive.

Turn on Sparse File Options

-S <n>

This command enables BRU to deal more intelligently with sparse files (files with many null bytes). When used in conjunction with the `-c` mode, this turns on automatic file compression for files that are larger than the specified size `<n>`. When extracting those files, the `-S` option should again be specified with the same `<n>`. When used in conjunction with the `-x` mode, seeks will be used to create blocks of null bytes in the output file, rather than actually writing null bytes.

Setting the Verbosity Level

-v

BRU normally tries to do its work silently, simply returning control to you once it has completed its task. Instead of guessing that all is going well, you may wish to see progress messages as BRU performs its tasks. If you are a new user of BRU, you may be interested in seeing the results of the commands that you have provided to BRU.

The `-v` option sets the verbosity (or “talkative”) level. There are actually four levels of verbosity available for selection: `-v`, `-vv`, `-vvv`, and `-vvvv`. The more `v`’s specified in the option, the more talkative BRU gets.

As an example of verbosity output, consider this example where the command `BRU -e -v` was entered:

```
1> BRU -e -v
```

e	4k of 6k [1]	kick
e	4k of 10k [1]	wbscreen.info
e	2k of 12k [1]	Expansion
e	4k of 16k [1]	Expansion/.info
e	2k of 18k [1]	lbin
e	110k of 128k [1]	lbin/BRU

The first field of the output contains the mode character, as a reminder of what mode is currently running (the -e or Estimate mode in this case). The next field gives the amount of space in kilobytes that this particular file uses in the archive. The next field ("of xxxk") gives a running total of the amount of data in the archive, including the current file. *NOTE:* there is an invisible header block of 2K, which is why the first value is 6K (2K header plus the 4K header block for the file Kick). The number inside the pair of square brackets is the current volume number. Finally, the name of the current file is listed.

Asking BRU to Wait for Confirmation **-w**

When the -w option is specified, for each file, BRU prints the file name, shows the action that it is about to take, and asks for confirmation that this action is OK to perform.

If you have a very long list of files, this option can take a long time. Therefore BRU provides a special response that tells BRU to continue without your prompts. This response is the g character. For example (responses are in bold):

```
1> BRU -e -w
kick: please confirm [y/n/g] y
  2k of 4k [1] kick
wbscreen.info: please confirm [y/n/g] y
  2k of 6k [1] wbscreen.info
Expansion: please confirm [y/n/g] y
  72k of 78k [1] Expansion
Expansion.info: please confirm [y/n/g] g
  68k of 146k [1] Expansion.info
```

Use LZW File Compression

-Z

If you specify the -Z option, BRU will use LZW file compression to make the final size of the archive smaller. Using this compression technique can result in space savings of 0% to over 90% depending on the kinds of files being stored. Typically, most files will compress about 30 to 50 percent, however sparse files containing lots of redundant data or zeros, such as large database files, may compress as much as 90% or more.

The default is to use 12-bit compression, but up to 16-bit compression can be used by specifying the -N <nbits> option, where n is the desired number of bits (12 to 16).

If you want to know how much each file is compressed, specify the -v option in conjunction with -Z and BRU will tell you what percentage of compression each file received. If you are simply cataloging an archive using the -t option, and the files are compressed, specifying the -Z will report the compressed size of each archived file, rather than the original size of the file.

File Selection Options

Selecting Files by Date

-n <date>

The -n option tells BRU that it should save or restore only files that are newer than some specific date (the -n stands for "newer than"). This option is useful when you are trying to set up a schedule for daily, weekly, or monthly backups.

For example, you might want to do a daily backup, and therefore specify the newer-than date as yesterday's date, at some specific time. Then only files that have been modified since that specific time yesterday will be copied into the archive. If you are frequently adding files or changing files, it is probably a very good idea to do daily backups of a filesystem. In addition to the daily backup, a weekly backup may be in order. On a selected day of the week, you can tell BRU to save only files that have been modified since the last weekly backup that you ran. Once a month you could do a full backup of all files and save these archives.

The frequency with which you should do backups, and the length of time for which you should preserve them, depends on how far back you might need to go to retrieve versions of files.

When using the `-n <date>` option, you may choose one of three possible formats:

`-n DD-MMM-YY[.HH:MM:SS]`

As indicated by the square brackets, the time specification is optional, and if present is separated from the day specification by a comma. If no time is specified, an hour of 00:00:00 is taken as the default.

DD	Is the day of the month. A leading zero (e.g. 03) to fill in the two digit field is necessary.
MMM	Is the name of the month, abbreviated to its first three characters.
YY	Is the last two digits of the current year.
HH	Specifies the hour.
MM	Specifies the minute.
SS	Specifies the seconds.

-n MM/DD/YY[,HH:MM:SS]

An alternate, equally acceptable format. In this case, the month is specified numerically.

-n MMDDHHMM[YY]

Another alternate, equally acceptable format. Again, all components are specified numerically.

Unconditional File Type Extraction -u <flags>

Normally in extraction, BRU will not overwrite an existing file with an older archive file of the same name. When used while restoring files, specifying the -u option causes files of the type specified by <flags> to be unconditionally selected regardless of modification times. Files which are not superseded will give warnings if the verbosity level is set at -vv or higher.

Characters for flags are:

- | | |
|---|---|
| a | Use any file, same as giving all other args |
| d | Use directories |
| f | Use regular files (same as "r") |
| r | Use regular files (same as "f") |

BRU with UNIX

The commands in this section are specifically for use with archives imported from UNIX systems.

Control Options

Control String

-# <str>

This option uses string <str> as a control string for the built in debugging system. This option provides information about the internal workings of BRU. It is only active in specially compiled debugging version.

Do Not Reset Access Time

-a

BRU normally changes the access times of a file while it reads it. The -a option instructs BRU not to reset these access times. Resetting these times prevents defeat of the mechanism used to track down and remove "dead" files that haven't been accessed recently.

Change the Owner of Extracted File

-C

This option tells BRU to change the owner (chown) and group of each extracted file to the owner uid and group gid of the current user. Normally BRU will restore the owner and group to those recorded in the archive. Use the -t -v option to see the owner and group of files stored in the archive.

Double Buffer

-D

The -D option causes BRU to use double buffering to the archive device. Depending upon hardware constraints, double buffering may dramatically increase the archive I/O rate, but may adversely affect error recovery.

Interaction Option

-I <option>

Using the interaction option allows you to run BRU from cron (cron runs programs at specified times and frequencies). If no interaction with the user is required, running from cron is no different than running directly from a terminal. However, when interaction is necessary there are two options: either terminate or find some way to communicate with the operator (or another program masquerading as the operator). The -B option provides for simple termination while the -I option provides for communication with an operator. BRU recognizes the following parameters for the -I option:

l,pathname	write verbosity info to pathname
q,fifo	write interaction queries to fifo
r,fifo	read interaction queries from fifo

You can use the device cycling feature (-f <path>) to specify that more than one archive device contains disks that are ready for a nightly backup. For example, you may load a set of disks into several drives and schedule cron to begin a daily backup at midnight. BRU will do as much work as it can without interaction, and then wait for you to provide any additional information in the morning.

Ignore Unresolved Links

-I

Normally, BRU reports problems with unresolved links (both regular and symbolic links). The -I option suppresses all such complaints.

Limit Directory Expansions to Same Mounted Filesystem

-m

This option instructs BRU not to cross mounted file system boundaries during expansion of explicitly named directories. This option applies only to directories named in files. It limits selection of directory descendants to those located on the same filesystem as the explicitly named directory. This option currently applies only to the -c and -e modes.

Select Files Owned by User

-o <user>

Sometimes you may want BRU to make your archive to include files owned by an individual user. This option lets you tell BRU for which user the archive is being created. BRU accepts three different forms of user identifications (<user>):

- an ASCII string that identifies the user name for which the archive is created. It must correspond to a user name in a password file. An example is:

```
BRU -c -o fred
```

- a pathname. By specifying a pathname to a file, BRU will assume that the owner of that file is the person for whom the archive is being created. An example is:

```
BRU -c -o work:fred [files]
```

- a decimal value.

Selection Options

Unconditional File Type Extraction **-u <flags>**

Normally in extraction, BRU will not overwrite an existing file with an older archive file of the same name. When used while restoring files, specifying the -u option causes files of the type specified by the flags to be unconditionally selected regardless of modification times. Files which are not superseded will give warnings if the verbosity level is set at -vv or higher.

Characters for flags are:

- | | |
|---|-----------------------------|
| b | use block special files |
| c | use character special files |
| l | use symbolic links |
| p | use fifos (named pipes) |

BRU Error Messages

With a properly configured Brutab file, BRU can recover from most common errors. In addition to those customized error indications in Brutab (where you tell BRU how your particular system indicates certain errors), BRU can generate many more error types. This section provides details on what those errors mean, what BRU was doing when the error happened, and possibly what you can do to correct the error.

Some errors cause BRU to stop, while other errors are recoverable. In general, BRU will avoid giving up and exiting in circumstances where it makes sense to attempt to continue. A warning usually results in an informational message.

For example, if you attempt to use a write protected disk during creation of an archive, BRU will warn you and allow you to reload volume one before continuing. When a volume is inserted out of order on multi-volume archive reads, BRU requests replacement of the out of sequence volume with the correct volume.

Error and warning messages in this section are given in alphabetical order. They are subdivided into three classes:

- Messages that start with a filename
- Messages that start with "warning —"
- Other messages

Conventions used for Error Messages

In the following descriptions, the notations filename, process, or string, indicate that BRU has printed a file name, process name, or appropriate string at that place in the error message.

When you see the notation 000, BRU has printed a numeric value in its place.

Messages Starting with Filename

filename: can't access for read

The specified file could not be accessed for read. This is usually because the read protection bit is not set.

filename: can't access for write

The specified file could not be accessed for write. This is usually because the write protection bit is not set.

filename: can't exec . . .

BRU could not exec the given file, for the reason that is given as part of the error message. Generally, this error occurs because the specified file does not exist, or is not executable by the user running BRU.

filename: can't link to filename

BRU had an error attempting to make a hard link between two files.

filename: can't open

BRU could not open the named file. The reason is given as part of the error message, and is usually because the read protection bit is not set.

filename: can't open archive

BRU cannot open the archive file. The reason is given as part of the error message.

filename: can't overwrite

The specified file could not be overwritten during extraction, and is usually because the write protection bit is not set.

filename: can't stat

The stat call or examine system call failed. Generally this indicates a permissions problem.

filename: could not make fifo

BRU was asked to create a fifo on a system that does not support fifos, and it could not even make a regular file of the same name.

filename: could not make symbolic link

BRU was unable to create a symbolic link.

filename: error making directory

BRU could not make a directory for some reason.

filename: error making node

BRU got an error while attempting to create a special file system node, such as a fifo, block special file, or character special file.

filename: error reading symbolic link

BRU could not read a symbolic link for some reason.

filename: line 000, obsolete brutab format

BRU detected an obsolete Brutab format while reading the Brutab file.

filename: media ejection failed

On systems that support ejection of archive media under software control, BRU may be configured to eject each media when it is finished with it. This error message indicates that BRU encountered some sort of error while attempting to eject the media.

filename: no file

The named file does not exist or part of the pathname is not searchable, and is usually because the read protection bit is not set.

filename: not deleted

BRU got some sort of error while attempting to delete a file.

filename: read error

BRU got a read error while reading a file from disk. This error should generally not occur. If it does, it usually indicates a hardware problem.

filename: symbolic links not supported

While reading during the differences mode (-d), BRU encountered a symbolic link on a system that does not support symbolic links.

filename: write error

BRU got an error while writing a file to disk. This error generally indicates a hardware problem or a full disk.

filename: warning — 000 additional link(s) added while archiving

While BRU was archiving a file, there were additional links made to it. These additional links may or may not have been archived.

filename: warning — 000 block checksum errors

While reading an archive, BRU detected the specified number of checksum errors in the specified file. This generally indicates a hardware problem.

filename: warning — 000 unresolved link(s)

While archiving a set of files, BRU detected that not all links to the specified file were found and archived. In other words, there is still another pathname that points to the same file, that does not appear in the archive.

filename: warning — block sequence error

BRU detected an inconsistency in the ordering of blocks returned by the archive device on a read. For example, BRU asked for blocks 11, 12, 13, 14, 15 and got blocks 11, 12, 14, 15, 16. This sort of error generally indicates a hardware problem, usually on tape drives.

filename: warning — close error on archive

BRU got some sort of error while attempting to close the archive file.

filename: warning — compressed version was larger, stored uncompressed

When file compression is utilized with the -Z control option, BRU will check to ensure that the compressed version of the file uses fewer archive blocks than the uncompressed version. If the compressed version does not result in any savings in archive space, the uncompressed version will be archived instead. If the verbosity level is 4 or greater, BRU will give warnings for those files for which compression was not effective in saving archive space.

filename: warning — compression failed, stored uncompressed

While attempting to compress a file for storage, BRU got some sort of error (such as overflowing the filesystem temporary space for example). BRU could not generate the compressed version of the file and therefore stored it uncompressed.

filename: warning — decompression failed, not extracted

While attempting to extract a file that was stored in compressed format, BRU encountered some sort of error that it could not recover from. In this case, the file is not extracted and this warning message is issued.

filename: warning — error setting mode

BRU got an error while attempting to set the mode of a file.

filename: warning — error setting times

BRU got an error while attempting to set the date or time of a file.

filename: warning — file close error

BRU got an error while closing the named file.

filename: warning — file grew while archiving

The specified file grew in length while BRU was in the process of reading it. If BRU was creating an archive at the time, the archived file is truncated to the size it was originally, when BRU started archiving it. This is the same size that is recorded in the file header block. This sort of warning is commonly seen for log or audit files, to which information is appended periodically. It can generally be avoided by only backing up the system in single user mode, but this is seldom worth the downtime.

filename: warning — file was truncated

The specified file was truncated while BRU was in the process of reading it. If BRU was creating an archive at the time, the archived file is padded with sufficient null characters to bring it back to the size it was originally, when BRU started archiving it. This is the same size that is recorded in the file header block.

filename: warning — lost linkage

BRU could not preserve the linkage of two files. This message is generally seen when BRU runs out of memory while attempting to allocate memory internally to maintain the linkage information. In this case, the file will be archived as two separate, distinct files in the archive. Only the linkage information will be lost.

filename: warning — not found or not selected

The user specified a file on the command line that was not found, either on disk or in the archive.

filename: warning — not superseded

During extraction, the specified file in the archive was found to be older (an earlier date) than the current file on disk, so the extraction was not performed. This behavior is the default to avoid accidentally overwriting new files with old files from an archive. The behavior can be overridden with the `-u` option. Also, this message will only be seen if the verbosity level is 2 or greater, because it is common to use BRU to overlay an existing file tree, only replacing or restoring files that are missing or out of date. In this case, one does not generally wish to be deluged with warning messages about the files that were not extracted.

filename: warning — unrecoverable archive write error, some data lost

While creating an archive, BRU got an unrecoverable write error, and all or part of the I/O buffer was lost. This message indicates which file or files in the archive were affected by the data loss, one message per file.

Messages Starting with warning

These messages all start with the string "warning —".

warning — all data currently on filename will be destroyed

When the Brutab entry for a device includes the qfwrite boolean value, this message will be issued on the first write to the first volume placed in that device, and BRU will wait for confirmation to continue. In devices which might share both mounted and unmounted media, this prevents inadvertently overwriting media that may have been left in the device by mistake.

warning — archive read error at block 000

BRU got an unrecoverable error while attempting to read an archive. Whatever data was available at that location in the archive is not recoverable.

warning — archive write error at block 000

BRU got an unrecoverable error while attempting to write an archive. Whatever data was to be written at that location in the archive has been discarded. Proper corrective action depends upon the situation and the specific file within which the error occurred.

If this error occurs on the first block of an archive, the archive may be write protected, in which case the wperr parameter is probably set wrong in the Brutab file, or BRU would have issued a different warning message. Another possibility is that the I/O buffer size is too large for the given device. Experiment with a smaller I/O buffer size (see the -b <size> option) and set the maxbufsize parameter in the Brutab file as appropriate.

warning — assuming end of volume 000 (unknown size)

While reading or writing a volume of unknown size, BRU has encountered an unrecoverable read or write error before reaching the end of the archive. It may have actually reached the end of the volume, or it may have simply reached a bad

spot on the media, which it cannot skip. Since it does not know the size, it has no way of knowing the difference, hence the warning message. If no other warnings or errors occur, this warning is benign.

warning — attempt to change buffer size from 000 to 000 ignored (incompatible brutab entries)

While reading or writing an archive using multiple devices and not using the -b option, BRU detected inconsistent default buffer sizes between the devices, usually as a result of reading a bufsize parameter from the Brutab entry. The buffer size is not allowed to change between volumes of an archive. To avoid this warning message use the -b option to force a specific buffer size for all volumes.

warning — buffer size (000) exceeds system imposed limit (000) with double buffering

warning — buffer size automatically adjusted to 000

While attempting to set up double buffering, BRU was asked to use an I/O buffer size that resulted in the double buffering buffers exceeding the system imposed shared memory limits. The I/O buffer size was automatically adjusted downwards to the maximum size that the system could support.

warning — estimate mode ignores compression

This warning message results when both the -e and -Z options are given simultaneously. Because of the large overhead in compressing files, and because there is no way to determine the compression ratio without actually doing the compression, BRU cannot estimate how much archive space is required for an archive when compression is enabled. Therefore, the -e option ignores possible savings due to compression.

warning — extracted fifo filename as a regular file

BRU was asked to extract a fifo on a system that does not support fifo's, so it extracted the file as a regular file.

warning — file synchronization error; attempting recovery . . .

BRU was expecting to find a file header block while reading an archive, but instead found another type of block. This warning is normal if you start reading an archive at a volume other than the first volume, or skip a volume in the middle of reading an archive. BRU will scan each successive archive block looking for a file header block, and normal processing will resume from the first file header block found.

warning — found volume 000 expecting 000

When extracting files from sequential volumes, you have inserted a volume out of sequence. Remove the volume and replace it with the correct one.

warning — label string too big

A user specified label, as given with the -L option, exceeds the limit imposed by the BRU archive format. This limit is currently 63 characters.

warning — link of filename to filename, filename does not exist

BRU was asked to create a hard link to a file that does not exist.

warning — link of filename to filename, filename is a directory, no link made

BRU was asked to create a symbolic link to an existing file that is a directory, on a system that does not support symbolic links. Since hard links to directories are not allowed by BRU, this warning is issued and no link is made.

warning — may have to use -F option to read archive

BRU spends considerable time computing checksums for each block of data that it writes to the archive. Some users disable the checksum feature with the -F option when creating archives, although this is not recommended. The -F option must also be given when reading such an archive, and BRU will issue this warning when it detects archive input that appears not to have any checksums.

warning — media appears to be unformatted

When BRU gets an error on the first read or write to a particular volume and the error conditions match the values set in the Brutab entry for unformatted media in this device, BRU will issue this warning message before asking for a formatted media. This message may also appear when there is, for example, no disk present in the disk drive. Note that if the format boolean value is set for the device, this warning will be suppressed and BRU will attempt to format the media.

warning — media appears to be unformatted or write protected

This is a general error that might appear on the first attempt to read or write an archive volume which is unformatted, or on an attempt to write to an archive volume that is write protected.

warning — media appears to be write protected

When BRU gets an error on the first write to a particular volume and the error conditions match the values set in the Brutab entry for write protected media in this device, BRU will issue this warning message before asking for another media.

warning — missing archive header block; starting at volume 000

BRU could not find the archive header block for some reason. This warning is normal when starting to read an archive at some volume other than the first volume.

warning — premature end of volume 000

BRU hit an end of file before the size of the archive volume you specified was reached.

warning — using internal default device table

BRU could not find the Brutab file generally located in S:, or could not find a Brutab file specified by the BRUTAB environment variable. In this case, it will then use the internal Brutab, which may not be correct for the current system.

warning — volume not part of archive created datestring

The volume that you have given to BRU, although the correct volume number, has a different creation time than that of the archive being extracted at the present time. The actual creation date and time are reported with this warning message.

Other Messages

bad number of bits (000) for compression (max 000)

While reading a file from an archive, BRU found a file that was compressed with more bits than it was capable of dealing with. This error should never occur unless the archive has been corrupted or unless an attempt has been made to read an archive that was written on another machine with other than the default number of bits (12-bits). The default is for BRU to use 12-bit compression.

can't allocate 000 archive buffer

BRU could not allocate an I/O buffer of the requested size. The maximum size is generally hardware and operating system specific, and may also depend upon what other programs are running on the system. For corrective action, attempt to reduce the amount of real memory in use, or rerun with a smaller buffer size.

can't allocate 000 more bytes

BRU has run out of memory for some reason. This generally occurs when memory is very limited on the system due to hardware or cpu constraints.

compression initialization failed, -Z suppressed

BRU was unable to acquire sufficient memory to perform the requested file compression. Try reducing the number of bits of compression, using the -N <nbits> option.

date conversion error: string

BRU could not convert the specified string into a date. This generally occurs due to a mistyped date string or not following the allowed syntax rules for date strings.

don't understand -I option string

The given *string* was not recognized as a valid argument for the interaction option -I.

double buffer child died, status 000

The child process used for double buffering died unexpectedly with the given status.

double buffer child error 000

The child process used for double buffering got some sort of fatal error that it was able to recognize as unrecoverable.

inconsistent wait status %o

The wait system call returned a value that BRU was not able to understand. Generally this error should not occur, but if it does, it probably indicates a kernel bug.

interaction needed, aborted by -B option

BRU was run with the -B option, indicating that no user was present and that the desired behavior, if interaction was necessary, was to abort.

internal bug in routine function

BRU has detected some sort of internal bug. This sort of error should not occur. If it does, it may possibly be a real bug in BRU or a hardware or software problem.

media size smaller than I/O buffer size!

The media size is inconsistent with the I/O buffer size. The media size must be at least as large as the I/O buffer. In other words, it is not possible to write a 1MB I/O buffer to a media that is only 500K.

need more than 000 shared memory segments

BRU was not able to allocate enough shared memory segments to properly set up double buffering.

no default device in brutab file, use -f option

BRU could not find any default device in the Brutab file.

no double buffer child to reap

While waiting for the double buffer child to exit, the wait system call failed for some reason that was unexpected in the parent process.

pathname filename too big

The pathname exceeds the maximum size which is permitted by the BRU archive format. This is currently 127 characters. This restriction may be lifted in a future version of BRU. For now it is suggested to archive the file using only part of the pathname, or reduce the number of intervening directories from the root directory.

pathname too big, lost filename

While building a file tree, BRU has created a path that exceeds the specified maximum pathname size (currently 127 characters).

problem setting up double buffering

BRU encountered a problem setting up the double buffering. Sometimes reducing the I/O buffer size will cure the problem.

process: fatal error; stopped by signal 000

The child process that BRU was waiting for was stopped by the specified signal.

process: fatal error; terminated by signal 000

The child process that BRU was waiting for was terminated by the specified signal.

seek error on archive

BRU got an unrecoverable seek error on an archive file.

specify mode (`—cdeghitx`)

BRU was run without specifying any sort of major mode.

Rerun with one or more mode flags specified.

support for obsolete brutab format not compiled in

Support for the original Brutab format, now obsolete, has not been compiled into this particular version of BRU, but BRU was asked to read an obsolete format Brutab file. Convert your Brutab file to the new format.

unterminated character class

BRU detected a Shell-style wildcard file specification that was syntactically incorrect.

Appendix D. Fountain

The Fountain utility, in the System drawer of the Extras2.0 disk, manages the installation of Intellifont® outline fonts onto your Amiga. The outline fonts are included on the AmigaFonts2.0 disk, along with the standard Amiga bitmap fonts. In general, Fountain works best on computers equipped with hard disks, although it is possible to use it with floppy disk systems.

Traditionally, the Amiga has used bitmap fonts. For each font style supplied on disk, there is a separate file containing the data used to produce each available size of that style. For instance, the Helvetica directory contains the following files: 9, 11, 13, 15, 18 and 24. Those files store the data used to create the corresponding Helvetica type sizes. Some application programs include conversion programs that allow you to use other sizes of the font. For instance, if you want to use Helvetica 20, the program will take the next closest Helvetica size and enlarge or reduce it to the appropriate size.

Outline fonts do not have separate files for each point size. Instead, the computer uses a mathematical formula to convert the basic font to whatever size is needed. While it takes slightly longer to access an outline font than a bitmap font, you do not need to store all the individual font files on your disks. Also, outline fonts appear on the screen just as they will appear on your printout, regardless of the type of printer you use. You no longer need to use a postscript printer to get high-quality output. Of course, the sizes you use on your screen are generally much smaller than the sizes you print.

Installing Fountain

If you have just brought a new Amiga computer with a hard drive, the program will already be installed in the System2.0 partition. If you are upgrading your software the UpdateWB program on the 2.0Install disk installs the files in the appropriate directories. If you need to install Fountain manually, please read the following section to learn how to install Fountain on your computer.

On a hard disk system:

You need to copy the contents of the AmigaFonts2.0 disk to your System2.0 (SYS:) partition. To do this, follow the steps below:

1. *Insert the AmigaFonts2.0 disk into the floppy disk drive.*
2. *Open the Shell icon in the System2.0 disk window.*
3. *Use the COPY command to copy the contents of the AmigaFonts2.0 disk to the System2.0 partition. Type:*

`COPY AmigaFonts2.0: Sys: ALL`

This will copy all the files on the AmigaFonts2.0 disk to the appropriate directories on System2.0.

NOTE: This will replace the standard diskfont.library file with a new outline fonts diskfont.library. You should not need the standard diskfont.library as the new library handles both bitmap and outline fonts. If for some reason you wish to use the standard library, you can restore it from your original Workbench2.0 disk.

On a floppy disk system:

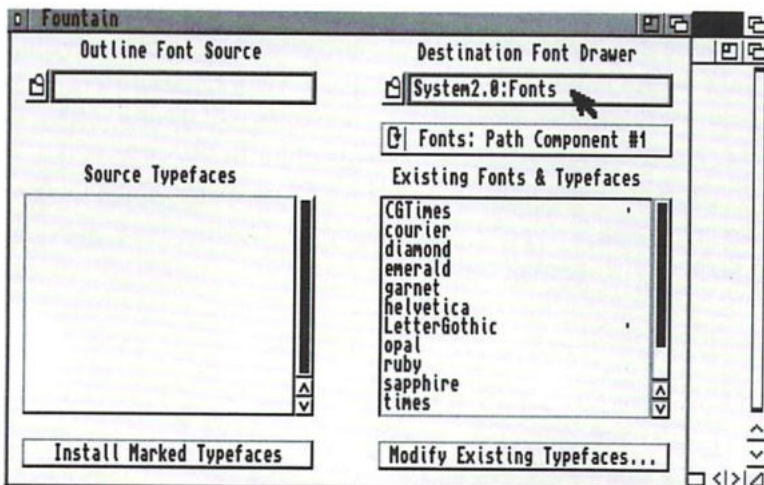
To use Fountain with a floppy disk system, we recommend you have at least two disk drives, and at least one megabyte of RAM. Boot your system with the Workbench2.0 disk, or another bootable application disk, and keep your AmigaFonts2.0 disk in the second drive. You will have to add an ASSIGN statement to the User-Startup file of your bootable disk telling the system that the Fonts directory is on the AmigaFonts2.0 disk. Open the User-Startup file, and add the following line:

```
ASSIGN FONTS: AmigaFonts2.0: DEFER ADD
```

The DEFER option tells ASSIGN not to request the AmigaFonts2.0 disk until the system needs it. Your system will then access the AmigaFonts2.0 disk whenever it needs to use the Fountain program or a font file.

Using Fountain

Double-click on the Fountain icon, and a window appears:



NOTE: If you press Help while in the Fountain program, instruction windows will appear to explain the gadgets in the window.

This window allows you to install additional fonts (aside from the fonts installed during the installation process) in one of your fonts directories. Fountain understands two typeface formats: Amiga Compugraphic® font disks and standard Compugraphic disks containing FAIS files. In the case of standard Compugraphic disks, you will have to first use an MS-DOS handler utility or Amiga Bridgeboard™ to convert the files to AmigaDOS format, since they are in MS-DOS format. Once the fonts are properly installed, they will be available to any application programs that use them.

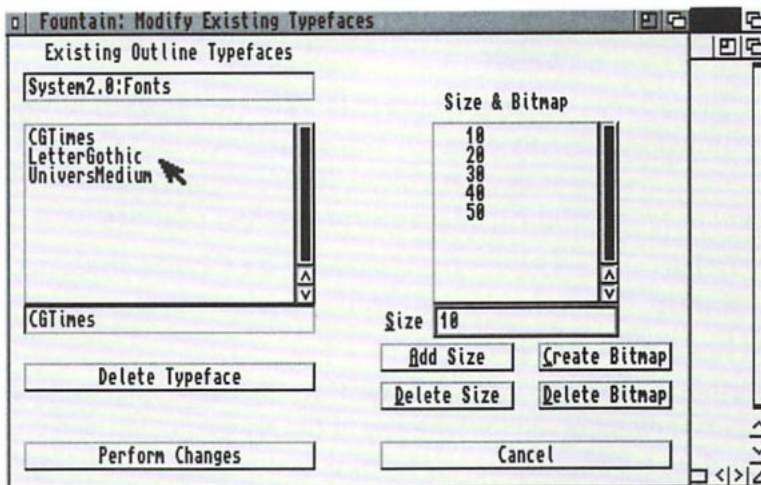
You must specify the complete path to the disk containing the fonts. To do this, either type the path in the Outline Font Source text gadget, or select the file folder gadget next to the text gadget to open a file requester. (Since you only need to specify a directory, the file text gadget is ignored in this requester.) Once the full path has been specified, Fountain will then display the available outline fonts in the Source Typefaces scroll gadget.

The typefaces will be installed in the drawer displayed in the Destination Font Drawer text gadget. If you have more than one fonts directory set up in your assign path, you can use the Fonts: Path Component cycle gadget to switch between the directories. You can also type in a non-assigned directory. (In this case, the cycle gadget will display Not in FONTS: Path.) The contents of the Destination Font Drawer are shown in the Existing Fonts & Typefaces scroll gadget. Outline fonts are indicated with a small bullet (·).

Select each typeface that you want to install by clicking on it. A plus (+) sign will appear. Then select the Installing New Typefaces gadget. These typefaces will be copied to the Destination Font Drawer. While the fonts are being installed, the minute hand in the wait pointer indicates approximately how much time is left in the process.

Whenever you install a typeface, the fountain environment variable ENV:Sys/Fountain is used to initialize the list of sizes that will typically be presented by applications in their font menus. You can change these sizes by selecting the Modify Existing Typefaces gadget. This will open a new window:

For more information on environment variables, see Chapter 7, "Using AmigaDOS."



The outline typefaces are listed in the Existing Outline Typefaces scroll gadget on the left side of the window. The path to the directory containing the fonts is shown in the display box above the scroll gadget. This directory cannot be changed from within the Modify Existing Typefaces window. (If you need to change it, select Cancel to return to the original Fountain window.) To select a typeface to modify, click on it. The selected typeface appears in the display box underneath the scroll gadget.

The available sizes for the selected typeface, as determined by the Fountain environment variable, are shown in the Size & Bitmap scroll gadget. The smaller sizes are appropriate for document text, while the larger sizes are often used for presentation graphics.

You can create new sizes for the typeface or create a bitmap file for a specific size by using the gadgets in the lower right corner of the window. These gadgets are described below:

Add Size	Enter the desired size into the Size display box, then choose the Add Size gadget.
Delete Size	Select the desired size or enter it into the Size display box, then choose the Delete Size gadget.
Create Bitmap	Creates a bitmap for the size displayed in the Size gadget. A bitmap file takes disk space but is quicker to load. If you routinely use one size, you may want to create a bitmap for it.
Delete Bitmap	Deletes the bitmap file for the size displayed. The size will still be available as an outline font.

Keyboard shortcut: Instead of selecting the above gadgets, you can simply press the initial letter of the gadget (underlined on the display). For instance, pressing A is the same as selecting the Add Size gadget.

To save your changes you must select the Perform Changes gadget. The Cancel gadget returns you to the original Fountain window. If you select the close gadget, you will close both Fountain windows and exit the program.

Changing Environment Variables

Fountain uses two environment variables to store specifications about your outline fonts: Fountain and Diskfont. The Fountain environment variable, stored in ENV:Sys/Fountain, is used to create the list of sizes that will typically be presented by applications in their font menus. By default these sizes are 15, 30, 45, 60 and 75. If you find yourself always using different sizes, you may want to change this variable. To do so, use a text editor and save the file to SYS:Prefs/Env-Archive/Sys/Fountain. The file should include a list of sizes saved in ASCII format. The maximum number of sizes permissible is 20.

The Diskfont variable, stored in ENV:Sys/Diskfont, specifies the parameters used by the diskfont library when it converts an outline typeface into an Amiga graphics font. The format of the variable is:

```
XDPI/N, YDPI/N, XDOTP/N, YDOTP/N
```

The XDPI and YDPI parameters adjust the aspect ratio. By default, the ratio is 1:1. If you know that the fonts will be used in the Hires display mode, you can adjust the aspect ratio appropriately by changing the XDPI value to 100 and the YDPI value to 50. To do so, use a text editor and save the file to SYS:Prefs/Env-Archive/Sys/Diskfont. Your file would look something like this:

```
XDPI 100 YDPI 50
```

If XDPI is specified, YDPI must also be specified.

The XDOTP and YDOTP parameters control the dot size percentage — the space a dot fills in relation to the screen resolution. The default value for both XDOTP and YDOTP is 100. This means that a dot fills the same size as implied by the resolution. There should be no need to change this default. If XDOTP is specified, YDOTP must also be specified.

NOTE: Very large or very small values of XDOTP or YDOTP are required before you see a difference.

Glossary

This glossary provides definitions of terms selected from the *Introducing the Amiga* and *Using the System Software* manuals.

absolute path

The explicit identification of a file or directory—one that includes the device or partition name and any directories that lead to the file.

acceleration

An option, selected through the Input editor, which causes the pointer movement to increase as the mouse is moved at a constant speed. Acceleration provides a higher degree of control for small mouse movements and less control, but greater mouse speed, for large movements.

action gadget

A box in a window that lets you choose an operation to be performed in the window by selecting the box. Common action gadgets are Save, Continue, and Cancel.

active

Currently selected, used in reference to the selected Workbench window.

address

An identifying number assigned to a device attached to the Amiga. For example, each SCSI device attached to your Amiga needs a separate address. The Amiga uses the address to locate the device.

alias

An alternative name for an AmigaDOS command or command string, specified with the ALIAS command.

AmigaDOS

The disk operating system (DOS) used by Amiga computers. A disk operating system provides the basic functions of the computer.

application

Instructions that tell the Amiga how to perform specific tasks, such as those required by a word processor, database, or video titler.

archive

1. {n} A backup copy of a file or files.
2. {v} To copy files to disk or tape for backup purposes.

argument

An additional piece of information, such as a filename, value, or option, included along with a command. This information determines the exact action of the command.

argument passing

Specifying, on the command line, parameters for a program or command to follow.

ASCII (American Standard Code for Information Interchange)

A standardized format for text that allows the exchange of information between different types of computers.

assign

To link a directory name to a logical device name, with the ASSIGN command, so that programs that use that directory can look for one device name rather than having to search through several volumes for the directory. For instance, the RAM:T directory is commonly assigned to the device name T:.

attributes

A series of flags stored with every file. Attributes indicate file type and control the operations (read, write, delete, etc.) permissible on the file. Also called *protection bits*.

autoscroll

To automatically move a screen when the pointer reaches the edges of the viewable area.

background process

A program that is started from the Shell with the RUN command. The program does not take over the Shell but is run in the "background."

backup

A copy of a file on disk or tape used to replace lost data.

back up

To make a backup copy.

bad block

A faulty area on a disk that cannot properly store information. Hard disk utilities often have provisions for "mapping out" (finding and marking as unusable) bad blocks detected on a hard disk.

baud rate

The speed at which a device receives or transmits information in serial communication. Roughly equivalent to bits per second.

binary

The base-2 number system which only uses the digits 0 and 1.

bit

1. A single binary digit (1 or 0).
2. A flag that has only two possible states (for example, deletable or undeletable).

block

1. A contiguous series of bytes (usually 512) treated as a single logical unit in RAM or permanent storage media.
2. A flag that has only two possible states (for example, deletable or undeletable).

boolean

Having two possible states: on or off, true or false, yes or no.

boot

To read the information needed to start the system from a storage device, such as a floppy or hard disk, into the computer's memory. Also refers to items used in this process: the *boot* disk. (See *reboot*.)

bootable

Refers to a device from which the Amiga can boot. A bootable disk must contain all the system files needed for the computer to start operation.

boot priority

A value assigned to each device to determine the order in which devices are checked to determine the boot device.

Bridgeboard

An expansion board made by Commodore that allows the Amiga to emulate PC and PC-compatible computers.

brush

An IFF graphics file, usually a section cut from a full-sized picture.

buffer

A temporary storage area in RAM.

bug

An error in software or hardware.

byte

A unit of memory consisting of eight bits, usually equivalent to one character.

cache

A temporary storage area (memory) used to improve system performance.

character pointer

In EDIT, the > symbol which is used to indicate the current position of the line window (line segment).

check box

A gadget that lets you turn an option on or off. When a check mark appears in the box, the option is selected, or on.

chip

A miniaturized electronic circuit, housed in a small, black, rectangular block edged by metal connector pins. A computer is made up of a variety of specialized chips.

Chip RAM

The area of RAM accessible to the Amiga's custom chip set. This memory is used for graphics and sound data. Also called *graphics memory*.

clear

1. To change a bit or flag to its 0, off or disabled state. Opposite of *set*.
2. To erase a screen or window display.

CLI (Command Line Interface)

A means of communicating with a computer by issuing commands from the keyboard. The program to let you do this on the Amiga is called the Shell. Before the Shell was available, the program used was called the CLI.

click

To press and release a mouse button.

close

To remove a window from the screen.

close gadget

A gadget which may appear in the upper left corner of a window to allow you to close the window.

cold reboot

To reset the Amiga by turning the power off, waiting 20 seconds, then restoring power.

color correction

A printing option, selected through the PrinterGfx editor, that tries to better match the colors of a printout to the colors on the screen.

command

A statement given to the Amiga to perform a task or achieve a result.

command history

A feature of the Shell which allows you to recall previously entered command lines by using the cursor keys.

command line

The line on which commands and their arguments are typed. Also, all the information that has been typed on the line.

compress

To make files smaller. Commonly used in backup programs so that you can fit more files on a floppy disk.

condition flag

A variable that contains a return code value indicating the success or failure of command execution.

console window

A window used for the input and output of text.

contiguous

Continuous; consisting of a series of adjacent items. *Contiguous* memory is a block of memory.

Control-key combination

A key combination that performs a special function, entered by holding down Ctrl while pressing another key on the keyboard. Some Control-key combinations are executed as soon as they are pressed, such as when Ctrl-C is used to abort the execution of an AmigaDOS command. Some produce a reversed character image and have no immediate effect.

controller

A hardware device, such as a hard disk controller, which acts as an interface between the computer and a peripheral.

coprocessor

A separate processor chip that assists the CPU by performing specific tasks, such as mathematical computations or rapid data transfer.

copy and paste

The act of copying a block of text to a new location within a console window.

CPU (Central Processing Unit)

The "brain" of a computer; the integrated circuit chip primarily responsible for executing the instructions in a program.

current directory

The current location in the directory structure. The directory AmigaDOS will use as the default directory to operate within, if no other directory is specified.

cursor

A highlighted rectangle on the screen used to indicate text position.

cycle gadget

A gadget for selecting one of several options. One option is displayed at a time, and as the gadget is selected, the other options become visible. The displayed option is the selected option.

cylinder

A logical division of a magnetic storage disk. Amiga 3.5 inch floppy disks are divided into 80 cylinders during the formatting process.

data

A collection of information.

dead key

A key, or key combination, which modifies the output of the next key to be pressed. For instance, on an American keyboard, Alt-H will superimpose a caret (^) symbol over the next key to be pressed. Alt-H is a dead key combination.

debug

To find and fix mistakes in software or hardware.

default

A value or action assumed if you have not specified anything.

Default Tool

A tool specified in the project icon's Information window. When the project icon is opened, the Default Tool is automatically loaded and run.

delete

To erase or discard a file, buffer, or other stored item.

delimiter

A character that marks the beginning and end of a string.

density

The number of dots per inch. Many printers support several print densities. Usually, the higher the density, the clearer the printout will be.

depth gadget

A gadget which may appear in the upper right corner of a window or screen for moving that window or screen in front of or behind other windows or screens. This is sometimes referred to as depth adjusting.

destination

The device, directory or file that is receiving information. For instance, in EDIT, the file that the revised text is being sent to is the destination file.

device

A physical mechanism, such as a printer or disk drive, or a software entity (logical device), such as CON: or NIL:, used as a source or destination for information.

device handler

Files that act as intermediate stages between AmigaDOS and physical devices, such as the Port-handler file in the L: directory which handles the interface for the PAR:, SER:, and PRT: devices.

device name

A short name, such as DF0:, FH2:, or PRT:, that identifies a particular hardware or software device. Device names must end in a colon (:).

directory

A subdivision in a computer's filing system used to organize files and other directories (subdirectories). Directories are represented on the Workbench as drawer icons.

disk

A medium for mass storage of computer data. Most computer disks store information magnetically; optical (laser-read) disks are also used.

disk drive

A storage device that reads and writes data from and to a storage disk, such as a floppy disk.

disk operating system

Software, supplied on floppy or hard disk, that controls the basic functions of a computer.

display box

A rectangular box, usually under a scroll gadget or next to a selection gadget, that displays the current selection. You cannot edit a display box.

display mode

A name given to the number of horizontal and vertical pixels that make up the screen. For instance, a Hires display mode is 640 pixels wide and 200 pixels high (for NTSC machines).

dithering

1. Creating smoother color or grey-scale shading of screen or printed displays by alternating pixel color or density. The Preferences PrinterGfx editor provides several settings for automatic dithering of printed graphics.
2. Creating the illusion of a color by using a pattern of other colors. For instance, creating the illusion of purple, by alternating pixels of red and blue.

double-click

To press and release the selection button twice.

drag

To move an icon, window, gadget, or screen across the display by pointing to the object, holding down the selection button, and moving the mouse.

drag selection

The process of selecting several icons at once by holding down the selection button and using the mouse to draw a box around the icons you want to select. When you release the mouse button, all the icons in the box will be selected.

drawer

A subdivision of a disk storage area. A drawer corresponds to an AmigaDOS directory.

drive name

A name assigned to a floppy disk drive or hard disk, such as DF0: or FH2:.

dump

A printout of the image displayed on the screen.

echo

To print a text string to the screen with the ECHO command.

ECS (Enhanced Chip Set)

The upgraded versions of the Amiga's Agnus and Denise coprocessor chips. The Enhanced Chip Set offers new display modes (ECS modes) and expands existing graphics capabilities. Many of the benefits of the ECS are available only in conjunction with Version 2.0 of the Amiga operating system.

editor

A program that lets you create and/or modify certain types of files. The Amiga provides Preferences editors to change Prefs settings and the text editors ED, MEMacs, and EDIT for changing text files.

environment variable

A variable used by AmigaDOS to represent a string or a value. Environment variables are commonly used in scripts and are supported by various programs, such as the Shell and More.

error code

A number identifying an error that has occurred during execution of a command or program.

escape sequence

A sequence of characters, beginning with the Escape character, that will perform a special function when entered on a command line or printed as part of a string. Escape sequences are typically used to change the appearance of text in a console window or to alter the style of type used by a printer.

execute

To carry out the instructions in a command line, program or script.

extended selection

The process of selecting several icons at once by holding down Shift while selecting each icon with the mouse.

extraction

In BRU, the process of restoring archived files.

FastFileSystem (FFS)

An enhanced Amiga file system usable with both floppy and hard disks. A volume is formatted as either FFS or OldFileSystem (OFS).

Fast RAM

General memory used by programs and data.

fatal

Describes an error serious enough to halt the process which caused it.

field

The screen area behind the text under a Workbench icon. The color of the field can be changed with the Font editor.

file

An organized collection of data referred to by a name.

file compression

See *compress*.

file system

The method of organizing stored files and the software for reading and writing them. The Amiga's native file systems are the FastFileSystem (FFS) and the OldFileSystem (OFS).

flag

A status indicator variable with a limited number of possible states.

floppy disk

A removable magnetic storage medium. The Amiga uses 3.5 inch, double-sided, double-density floppy disks in a rigid plastic case; they can store approximately 900,000 bytes (880K) of information.

font

A particular design of a set of letters, symbols, and numbers used for text display, such as Topaz and Helvetica. Fonts are usually available in several sizes, defined in points (10 point, 12 point, etc.).

format

1. To prepare a disk for use with the Amiga. Formatting a disk erases all previously stored data.
2. A way of describing the proper syntax for AmigaDOS commands.

fragmentation

An uneven distribution of data on a disk, causing the computer to look in different locations on the disk to find the information.

function keys

Keys at the top of the Amiga keyboard, labeled F1 to F10, that can be programmed to perform special tasks.

gadget

Any of various programmed graphic images which may appear in a window, requester or screen, that can be manipulated with the mouse to perform a certain function. Each gadget is of a specific type and performs a specific action. When selected, gadgets may appear to sink into the screen.

genlock

A device that allows Amiga graphics to be synchronized and combined with external video sources, such as a TV or VCR.

ghosting

Displaying menu or gadget items on the screen less distinctly than normal to indicate that they are currently unavailable.

global

Effective in all processes. Opposite of *local*.

graphics memory

See *Chip RAM*.

GUI (Graphical User Interface)

A visually-oriented system allowing you to tell a computer what to do by manipulating graphic symbols rather than by typing in commands. The Workbench is the Amiga GUI.

HAM (Hold And Modify)

An Amiga graphics mode that allows all the Amiga's 4096 colors to be displayed on the screen at the same time.

handshaking

The electronic protocol required for communication between two computing devices.

hard disk

A high-speed, large-capacity mass-storage device, from which the disks usually cannot be removed. Often called a hard drive or hard disk drive.

hexadecimal

The base-16 number system often used in computer programming. Hexadecimal (hex) numbers use the letters A-F as well as the digits 0-9, and must be preceded by the characters 0x or #x to be recognized as such by AmigaDOS.

hex number

See *hexadecimal*.

hierarchical

A term used to describe the multi-leveled AmigaDOS file structure in which directories can contain other directories and/or files.

history buffer

A section of memory that stores the most recent commands for a given Shell.

hold down

To continually press a mouse button until instructed to release it.

hot key

A key or key combination used by Commodore Exchange programs to open a hidden window.

icon

An image appearing on the screen to represent a disk, drawer, project or tool. Icons can be moved and selected with the mouse to allow you to work with the items they represent.

IFF (Interchange File Format)

The standardized format in which the Amiga stores picture and sound data.

.info file

A file containing the image and position data for an icon; pronounced "dot-info."

initialize

A synonym for format.

input buffer

An area of memory used during serial communication to hold incoming information.

interlace

An aspect of some Amiga display modes that doubles the vertical screen resolution.

Internal

Refers to an AmigaDOS command that is built into the Shell, rather than loaded from disk.

jumper

A small component on a circuit board that can be adjusted to one of several positions, used to control a certain aspect of the system configuration, such as how many disk drives have been installed or how much expansion memory has been added.

K (Kilobyte)

1024 bytes. Often abbreviated as KB.

key block

A number which identifies the block on the disk where a file is stored. Also known as a *file header block*.

keyboard shortcut

A method for performing a mouse action by pressing a key or key combination.

keymap

A file which determines the arrangement of characters on the keyboard and determines the meaning of each key. Different languages have different keymaps.

keyword

A word recognized by an AmigaDOS command as identifying an argument or specifying an option.

Kickstart

Software that is read from disk to boot the Amiga. Also refers to the portion of the operating system that is in ROM.

label

A point in a script to which execution may be directed to skip if certain conditions are met; specified with the LAB command.

library

A related set of functions and collections of data that can be shared by various programs. For instance, the Commodities.library in the LIBS: directory is used by all the Commodity Exchange programs.

line verification

In EDIT, the process of displaying a revised line after the command to change the line has been executed.

line window

In EDIT, a subsection of the current line.

link

A file that is a pointer to another file. When the original file is called, the linked file will be used.

local

Effective only in the process in which it was defined. Opposite of *global*.

low-level format

To prepare a new hard disk, making it possible for it to read and write information.

LUN (Logical unit number)

A value from 0 to 7 used as a secondary address. It is used when a device controls multiple devices, such as when a controller card with a SCSI address controls multiple hard disks. Each device attached to the controller must have a different LUN.

macro

A single command that represents a sequence of commands. Many editors and applications support the use of macros to facilitate commonly used command sequences.

MB (Megabyte)

1024K (1,048,576 bytes). Often abbreviated as M or Meg.

memory

The Amiga's internal storage circuitry which holds programs and data. The Amiga has both Chip (graphics) RAM, Fast (normal) RAM, and 512K of ROM memory. The amount of RAM memory limits the size and number of programs that can be operating within the Amiga at one time.

menu

A list of on-screen options, displayed by using the menu button, from which you can choose commands that control a program.

menu bar

The list of headings that appears across the top of the screen when the menu button is held down.

menu button

The right mouse button.

menu item

An option that appears in a menu. For example, New Drawer is the first menu item in the Workbench's Window menu.

modem

A device allowing serial communication over telephone lines.

module

An element of a program that performs a task of a limited scope, e.g., a module used to open a requester. A module may be used by a number of programs.

monitor

A video display terminal on which a computer's visual output is shown. There are many types of monitors; the Amiga's standard output uses an analog RGB color monitor to display both graphics and text.

motherboard

The main circuit board in a computer on which the CPU, main memory, etc. are located.

MountList

A text file in the DEVS: directory that contains information about devices that have been attached to or installed in the Amiga.

mouse

The device used to move the pointer on the screen and to communicate with the Amiga. Its buttons can be used for displaying menus, and for selecting and dragging icons, windows and screens.

multiprocessing

The ability to have more than one CPU chip in a computer functioning simultaneously, each executing its own processes, thus vastly improving overall performance. For instance, the CPU can perform calculations while another processor is drawing an object on the screen.

multiscan

A type of video monitor that can accept several different scan rates (types of video output).

multitasking

The ability to perform more than one operation, or task, at a time. The Amiga can have several independent programs running at once. For instance, you could simultaneously be displaying an animation, playing a sound file, communicating with another computer, and formatting a floppy disk.

nonproportional font

A font in which each character takes up an equal amount of space. For instance, an uppercase W is allotted the same amount of space as a lowercase l.

null string

An empty string. Null strings are commonly used in text editors to delete information. If you replace a word with a null string, the word is deleted.

offset

To shift or move over.

open

To make the selected object available for use. You open an icon by double-clicking on it or by selecting it then choosing the Open menu item from the Icons menu. When you open a disk or drawer icon, its contents are displayed. When you open a project or tool icon, a program is started.

output queue

In EDIT, the area of memory that is reserved to hold processed lines before writing them to the destination file.

overscan area

The normally unused area surrounding a standard-size screen. The Overscan editor allows you to expand your screen to fill this area.

overwrite

To write information to a file or disk, replacing any information that previously was stored there.

parallel

An interface port that transfers data one complete byte (8 bits) at a time, contrasted to a serial interface which sends a single bit at a time. The Amiga has an external parallel port to which a printer is often connected.

parent

The window or directory from which another window, directory, or file was generated. For instance, the Workbench window is the parent window of the disk windows. The Devs directory is the parent directory of the Printers directory.

parity

A method of detecting errors in serial communication by attaching an extra bit to bytes of data.

partition

A distinct section of a hard disk, used for data storage, which works as a physically separate device.

path

The series of device, directory, and subdirectory names that define the location of a file.

pattern

A set of characters shared by one or more file or directory names.

pattern matching

An AmigaDOS feature that lets you specify file and directory names by using wildcard characters. With wildcards, you can create search patterns that allow you to refer to a number of files that share a common text pattern without naming each file individually.

peripheral

An external hardware device connected to the Amiga.

pitch

The number of characters printed in a horizontal inch.

pixels

The dots of light that make up the Amiga screen display. A pixel is the smallest unit of display information on a given screen.

pointer

An image on the screen, usually arrow-shaped, that moves as you move the mouse. You use the pointer to select icons and gadgets and to choose menu items.

Preferences (Prefs)

A Workbench drawer containing editors that let you configure and customize your Amiga environment, such as changing the colors of your screen and setting the specifications for communication through the serial port.

printer driver

A program that enables the Amiga to communicate with your printer. A printer driver works as a translator between a computer and a printer, taking the information from the computer and presenting it to the printer in a format that the printer can understand.

priority

A variable determining the proportion of the Amiga's processing time that will be allotted to a given task. Each task has an independent priority. Task priority is set automatically but can be changed with the `CHANGETASKPRI` command.

process

A task that can communicate with AmigaDOS. Each process has a unique *process number*. Shell process numbers are usually displayed as part of the Shell prompt.

program

A series of instructions that tell the Amiga how to perform certain tasks. Applications and system software are programs.

project

A file in which information created or used by a tool is stored. For instance, files created with a text editor or paint program are projects.

prompt

A message or symbol, such as `1>`, that indicates that text input to the computer is possible.

protection bits

(See *attributes*.)

pseudo-icon

An icon that is displayed, when the Show All Files menu item is chosen, for an object that does not have a `.info` file.

pure

Describes a command or program that can be made resident. If a file is pure, the `p` attribute is set.

qualified string

In EDIT, an argument that pertains to a particular area of the current line, such as the beginning or end. For instance, a qualified string may specify that text should be replaced only if it appears at the beginning of the line. If it appears elsewhere in the line, it is left alone.

qualifier

1. A key, such as Shift, Ctrl, or Alt, that changes the Amiga's interpretation of a simultaneous or subsequent keystroke or mouse click. Commonly used with Commodity Exchange programs.
2. In EDIT, a character used with a command to designate where in the line the command should act. For instance, if a B qualifier is used with a command, the command will only work on the beginning of the line.

radio button

A circular gadget beside an option on a list. To select an option, select its radio button. You can only select one option from the list at a time.

RAM (Random Access Memory)

Part of the Amiga's internal memory that can be used for data storage and is directly accessible by the CPU. Applications are loaded into RAM from disk and use additional RAM to process and store data while the computer is on. Data in RAM is lost when the Amiga is rebooted or powered off.

Ram Disk

A section of RAM set aside to function as if it were a disk drive. This is much faster than a physical drive, since there are no mechanical elements. The Ram Disk is also known by the logical device name of RAM:.

read

To retrieve stored information.

Read Only

If disk status is Read Only, you can only look at the contents of the disk, you cannot alter them.

Read/Write

If disk status is Read/Write, you can both look at and alter the contents of the disk.

read/write head

A small, electromagnet that uses magnetic impulses to record or read data on a hard disk.

reboot

To reset the Amiga by pressing Ctrl, left Amiga, and right Amiga. This is roughly equivalent to turning the power off, then on again. Memory is reset. Also called *warm boot*.

redirect

To change the source or destination of a command's input or output from the default by using the special characters < or >.

relative path

In BRU, the path to a file or directory that does not include the device or partition name that leads to the file.

requester

A window that appears when the system needs a response from you. A requester contains action gadgets that give you a choice of continuing or aborting the operation in progress. To exit the requester, you must select one of the displayed gadgets.

resident

Describes a command or program that has been copied into memory, with the RESIDENT command, for quicker execution. Resident commands are specially set up to prevent reloading on subsequent executions. Only pure files can be made resident.

resolution

The number of pixels associated with a particular display mode. For example, a normal NTSC Hires screen has a resolution of 640 (horizontal) by 200 (vertical) pixels.

restore

To retrieve files that have been backed up or archived.

return code

A numerical value, generated upon execution of a command, to indicate its level of success. The number is 0 if the command was successful, and usually 5, 10, or 20 if there was a problem in executing the command. The return code value is assigned to the condition flag.

RGB (Red-Green-Blue)

A type of video signal in which the three primary color signals are sent separately. Standard Amiga output uses an RGB monitor.

ROM (Read Only Memory)

Permanent memory that is pre-programmed with system instructions and does not change. The contents of ROM are not affected by user commands or program operation.

root block

The area of a disk that contains the name of the disk and information pertaining to the disk layout. If the root block is erased, you cannot retrieve any information from the disk — it is effectively blank.

root directory

The main directory on a volume. The root directory is at the top of the filing hierarchy, and is created when a volume is formatted. All other directories on the volume exist within the root. The root directory is specified by the volume name followed by a colon.

scaling

Changing the size of an image during printing. Usually, a screen image is scaled down to a smaller size for printing, but you can also enlarge, or scale up, an image.

screen

An area of the display that shares the same video attributes, such as resolution and colors. Screens are always at least the full width of the viewable area.

script

A text file containing a series of commands that can be automatically executed to perform a complex or repetitive task. An example of a script is the Startup-sequence file executed when you boot your Amiga.

scroll

To move through the viewing area of a window.

scroll arrows

Gadgets which may appear in a window to allow you to move the viewing area continuously.

scroll bar

The highlighted area within the scroll box that can be dragged to display the hidden contents of a window. It changes in size to indicate the portion of the window that is currently visible.

scroll box

The shaded area within which the scroll bar can be dragged. You can click in the scroll box to move the scroll bar.

scroll gadget

A gadget which may appear in a window to let you move through a list of options or through the viewing area of a window. A scroll gadget is made up of the scroll bar, scroll box, and scroll arrows.

scrolling list

The options that appear inside a scroll gadget. If the list is longer than what can be displayed in the scroll gadget, you can use the scroll bar or scroll arrows to move (scroll) through the list.

SCSI (Small Computer System Interface)

A standard interface protocol for connecting peripherals, usually mass storage devices, to computer equipment. (Pronounced "scuzzy".)

search path

The list of directories that AmigaDOS uses when it is looking for a command. Directories are added or removed from the search path with the PATH command.

sector

The smallest unit of storage on a hard disk, usually equal to 512 bytes.

select

To choose an item to work with by pointing to it with the mouse, then pressing and releasing the selection button.

selection button

The left mouse button.

selection gadget

A gadget from which you can choose one of several displayed options, often used for colors.

serial

An interface port that transfers data one single bit at a time, contrasted to a parallel interface which sends one complete byte (eight bits) at a time. The Amiga has an external serial port to which a modem, MIDI interface, or printer is often connected.

set

To change a bit or flag to its on or enabled state. Opposite of clear.

Shell

The command line interface used to send typed commands to the Amiga. The Shell is a console window which supports many special features, such as command-line history, aliases, and copy and paste operations.

sizing gadget

A gadget which may appear in the lower right corner of a window to allow you to enlarge or shrink the size of the window.

slider gadget

A gadget from which you can select a value by dragging a bar through the gadget. As you move the slider bar, different values are displayed.

slider value

A number that appears next to a slider gadget to indicate the currently selected value.

slot

An internal receptacle with a certain number of connector pins into which an expansion board can be inserted.

smoothing

A printing option available in the PrinterGfx editor that attempts to eliminate, or smooth, jagged lines that can sometimes appear in printouts.

snapshot

To save the positions of a window and/or the icons within it.

socket

The receptacle, attached to a circuit board, into which a chip (such as a RAM chip) can be inserted.

source

A device, directory or file that is supplying information. For instance, when you copy a disk, the disk you are copying is the source disk. If you are using EDIT, the file you have read into memory and want to edit is the source file.

stack

A special area of RAM reserved by a program for temporary storage.

Startup-sequence

An AmigaDOS script file, executed when the Amiga is booted, that helps set up the hardware and directory systems.

stop bits

Extra bits added to signal the end of a character, used during serial communication.

streaming tape

A high-capacity, mass-storage device that uses a magnetic tape cartridge to hold data; generally used to back up large hard disks.

string

A piece of text treated as a single unit.

subdirectory

A directory that is within another directory; equivalent to a drawer within a drawer.

submenu

A secondary menu that appears when some menu items are highlighted. If a menu item produces a submenu, a >> symbol appears to the right of the menu item.

swap

To alternately place different floppy disks into the same drive, as when performing a single-drive disk copy.

switch

A command keyword that turns an option on or off. If the keyword is present, the option is used.

syntax

The rules for the proper arrangement of commands, keywords, and punctuation in a command line or programming language.

task

A software function spawned by a process. A program, command, or system operation that is being executed is a task.

Template

A way of specifying the proper syntax for an AmigaDOS command. You can display a command's Template by typing the command name followed by space and a question mark (?).

terminator

In EDIT, text that specifies that the end of a block has been reached.

text gadget

A rectangular box in which you can type information, such as a filename or command. Text gadgets are used by the Rename and Execute Command menu items, as well as many programs.

threshold

A PrinterGfx value related to color intensity. It determines which colors are printed as black and which are printed as white during black-and-white printing.

timestamp

The date and time associated with a file. This is usually the date and time when the file was created or last modified.

title bar

The top border of a screen or window, which commonly displays the name of the screen or window.

tool

A program that creates or uses data, such as a text editor or paint program.

Tool Type

An optional parameter that you can enter in an icon's Information window to control a program. For instance, if you enter the SECONDS Tool Type in the Clock's Information window, the Clock will display the seconds every time it is opened.

track

Divides a hard disk platter into concentric circles.

Trashcan

A directory for storing files that you want to delete.

type ahead

A feature of the Shell that lets you enter commands as a previous command's output is being displayed.

verbosity

In BRU, the mode in which the program becomes "talkative" and reports each action it takes as it is happening.

version

A number identifying a piece of software: *Workbench version 36.68*.

volume

A floppy disk or hard disk partition.

volume name

The name under the icon for a disk or partition. Renaming a disk changes its volume name.

wait pointer

An image of a stopwatch that appears in place of the normal pointer when the Workbench is busy and cannot accept further input.

wildcard

A symbol used in pattern matching to represent a range of possible values, such as when specifying filenames that all start or end with the same character. The question mark (?), for example, is used as a wildcard to match any single character.

window

A rectangular screen area that can accept or display information. A window has a title bar identifying it and may contain gadgets in its border.

Workbench

The Amiga's icon-based, graphical user interface.

Workbench2.0 disk

The floppy disk that contains the AmigaDOS Version 2.0 software.

write

To record data in memory or on a magnetic storage medium such as disk or tape.

write-enable

To allow information to be written onto a storage device. When a floppy disk is write-enabled, a small, plastic tab is covering the hole in the corner of the disk.

write-protect

To prevent information from being written onto a storage device. An individual file can be write-protected by clearing its w attribute. Floppy disks have a plastic tab which can be moved to write-protect the entire disk.

zoom gadget

A gadget which may appear in the upper right corner of a window to allow the window to alternate between two sizes.

Zorro

The name for the 100-pin expansion slot specification used by Amiga computers. Amiga 2000 and 3000 families contain Zorro II and Zorro III slots, respectively.

Index

A

- A2024 monitor, 4-7
- ABBREV, 10-91, 10-92
- about your hard disk, 6-1—6-3
- ABS, 10-92
- absolute path, C-17
- action gadgets, 2-21, 2-34, 2-39
- active, 1-15
- ADDBUFFERS, 8-7, 8-8
- adding a Tool Type, 4-2—4-4, 4-16, 4-17
- adding an ASSIGN statement to your
 - User-startup, 6-8—6-13
 - example ASSIGN statement, 6-11—6-13
- adding bad blocks to the bad blocks list,
 - 6-78—6-80
- ADDLIB, 10-92, 10-93
- AddMonitor, 4-6—4-10, 5-2
- ADDMONITOR, 8-8, 8-9
- ADDRESS, 10-50, 10-51, 10-93
- advanced options with partitioning,
 - 6-60—6-62
- alias, 7-41, 7-42
- ALIAS, 8-9, 8-10
- ALLOCMEM, 10-129, 10-130
- Amiga drive names, 1-28
- Amiga key, 1-41
- AmigaDOS command conventions,
 - 8-1—8-6
 - Format, 8-3, 8-4
 - Template, 8-4—8-6
- AmigaDOS command specifications,
 - 8-7—8-132
 - ADDBUFFERS, 8-7, 8-8
 - ADDMONITOR, 8-8, 8-9
 - ALIAS, 8-9, 8-10
 - ASK, 8-11
 - ASSIGN, 8-12—8-16
 - AUTOPOINT, 8-16, 8-17
 - AVAIL, 8-17, 8-18
 - BINDDRIVERS, 8-18
 - BINDMONITOR, 8-19
 - BLANKER, 8-20, 8-21
 - BREAK, 8-21, 8-22
 - CALCULATOR, 8-22
 - CD, 8-22, 8-23
 - CHANGETASKPRI, 8-24
 - CLOCK, 8-25, 8-26
 - CMD, 8-26, 8-27
 - COLORS, 8-27, 8-28
 - COPY, 8-28—8-30
 - CPU, 8-31—8-33
 - DATE, 8-33, 8-34
 - DELETE, 8-35, 8-36
 - DIR, 8-36—8-38
 - DISKCHANGE, 8-38, 8-39
 - DISKCOPY, 8-39, 8-40
 - DISKDOCTOR, 8-40—8-42
 - DISPLAY, 8-42—8-44
 - ECHO, 8-44, 8-45
 - ED, 8-45
 - EDIT, 8-45
 - ELSE, 8-46, 8-47
 - ENDCLI, 8-47
 - ENDIF, 8-47
 - ENDSHELL, 8-48
 - ENDSKIP, 8-48
 - EVAL, 8-49—8-52
 - EXCHANGE, 8-52
 - EXECUTE, 8-53—8-58
 - FAILAT, 8-58, 8-59
 - FAULT, 8-60
 - FILENOTE, 8-60, 8-61
 - FIXFONTS, 8-62
 - FKEY, 8-62, 8-63
 - FONT, 8-63, 8-64
 - FORMAT, 8-65

- GET, 8-66
- GETNV, 8-66, 8-67
- GRAPHICDUMP, 8-67
- ICONEDIT, 8-68
- ICONCONTROL, 8-68, 8-69
- ICONX, 8-69, 8-70
- IF, 8-70—8-72
- IHELP, 8-72, 8-73
- INFO, 8-73, 8-74
- INITPRINTER, 8-74
- INPUT, 8-75
- INSTALL, 8-76, 8-77
- IPREFS, 8-77
- JOIN, 8-77, 8-88
- KEYSHOW, 8-78
- LAB, 8-78
- LIST, 8-79—8-82
- LOADWB, 8-83
- LOCK, 8-83, 8-84
- MAKEDIR, 8-84, 8-85
- MAKELINK, 8-85
- MORE, 8-86
- MOUNT, 8-87
- NEWCLI, 8-88
- NEWSHELL, 8-88—8-90
- NOCAPSLOCK, 8-90
- NOFASTMEM, 8-91
- OVERSCAN, 8-91, 8-92
- PALETTE, 8-92
- PATH, 8-93, 8-94
- POINTER, 8-95
- PRINTER, 8-96
- PRINTERGFX, 8-96, 8-97
- PRINTFILES, 8-97
- PROMPT, 8-98, 8-99
- PROTECT, 8-99—8-101
- QUIT, 8-101, 8-102
- RELABEL, 8-102
- REMRAD, 8-103
- RENAME, 8-103, 8-104
- RESIDENT, 8-104—8-107
- RUN, 8-107, 8-108
- SAY, 8-108, 8-109
- SCREENMODE, 8-109, 8-110
- SEARCH, 8-110, 8-111
- SERIAL, 8-112
- SET, 8-113
- SETCLOCK, 8-114
- SETDATE, 8-115
- SETENV, 8-116
- SETFONT, 8-117
- SEMAP, 8-118
- SETPATCH, 8-119
- SKIP, 8-119, 8-120
- SORT, 8-121
- STACK, 8-122
- STATUS, 8-123
- TIME, 8-124
- TYPE, 8-124, 8-125
- UNALIAS, 8-125
- UNSET, 8-125
- UNSETNV, 8-126
- VERSION, 8-126, 8-127
- WAIT, 8-127
- WBCONFIG, 8-128
- WBPATTERN, 8-129, 8-130
- WHICH, 8-130, 8-131
- WHY, 8-131, 8-132
- AmigaDOS error messages, 8-133—8-136
- AmigaDOS reference, 8-1
 - application software, 1-1, 1-42, 1-43, 2-3
 - archive, 6-14, C-1
- AREXX Built-in Function Library, 10-90, 10-91
- AREXX Built-in Functions, 10-85, 10-87, 10-91—10-128
 - ABBREV, 10-91, 10-92
 - ABS, 10-92
 - ADDLIB, 10-92, 10-93
 - ADDRESS, 10-93
 - ARG, 10-93

- B2C, 10-94
- BITAND, 10-94
- BITCHG, 10-94
- BITCLR, 10-95
- BITCOMP, 10-95
- BITOR, 10-95
- BITSET, 10-96
- BITTST, 10-96
- BITXOR, 10-96
- Built-in Functions-examples, 10-125—
10-128
- C2B, 10-97
- C2D, 10-97
- C2X, 10-97
- CENTER or CENTRE, 10-98
- CLOSE, 10-98
- COMPRESS, 10-98
- COMPARE, 10-99
- COPIES, 10-99
- D2C, 10-99
- D2X, 10-100
- DATE, 10-100, 10-101
- DATATYPE, 10-101, 10-102
- DELSTR, 10-102
- DELWORD, 10-102
- DIGITS, 10-102
- EOF, 10-103
- ERRORTXT, 10-103
- EXISTS, 10-103
- EXPORT, 10-103, 10-104
- FORM, 10-104
- FIND, 10-104
- FREESPACE, 10-104, 10-105
- FUZZ, 10-105
- GETCLIP, 10-105
- GETSPACE, 10-106
- HASH, 10-106
- IMPORT, 10-106
- INDEX, 10-107
- INSERT, 10-107
- LASTPOS, 10-107, 10-108
- LEFT, 10-108
- LENGTH, 10-108
- LINES, 10-108
- MAX, 10-109
- MIN, 10-109
- OPEN, 10-109
- OVERLAY, 10-110
- POS, 10-110
- PRAGMA, 10-110—10-112
- RANDOM, 10-112
- RANDU, 10-112, 10-113
- READCH, 10-113
- READLN, 10-113
- REMLIB, 10-113, 10-114
- REVERSE, 10-114
- RIGHT, 10-114
- SEEK, 10-114
- SETCLIP, 10-115
- SHOW, 10-115
- SIGN, 10-116
- SOURCELINE, 10-116
- SPACE, 10-116
- STORAGE, 10-117
- STRIP, 10-117, 10-118
- SUBSTR, 10-118
- SUBWORD, 10-119
- SYMBOL, 10-118, 10-119
- TIME, 10-119
- TRACE, 10-120
- TRANSLATE, 10-120
- TRIM, 10-121
- TRUNC, 10-121
- UPPER, 10-121
- VALUE, 10-122
- VERIFY, 10-122
- WORD, 10-122
- WORDINDEX, 10-123
- WORDLENGTH, 10-123
- WORDS, 10-123
- WRITECH, 10-123
- WRITELN, 10-124

- X2C, 10-124
- X2D, 10-124
- XRANGE, 10-124, 10-125
- AREXX Commands, 10-74—10-82
 - command clauses, 10-74, 10-75
 - command interface, the, 10-78, 10-79
 - command inhibition, 10-82
 - host address, the, 10-75—10-78
 - using AREXX with command shells, 10-81
 - using commands in macro programs, 10-79, 10-80
- AREXX Error Messages, 10-157—10-166
- AREXX functions, 10-82—10-90
 - Built-in functions, 10-85, 10-87
 - Clip List, the, 10-89, 10-90
 - External AREXX programs, 10-86
 - External Function Libraries, 10-88, 10-89
 - Function Libraries and Function Hosts, 10-86
 - Internal functions, 10-85, 10-87
 - Library List, the, 10-83, 10-84
 - syntax and search order, 10-84—10-89
- AREXX Instructions, 10-18—10-26, 10-50—10-73
 - ADDRESS, 10-50, 10-51
 - ARG, 10-52, 10-64
 - BREAK, 10-52
 - BREAK—C, 10-71
 - BREAK—D, 10-71
 - BREAK—E, 10-72
 - BREAK—F, 10-72
 - CALL, 10-53
 - DIGITS, 10-65
 - DO, 10-53—10-56
 - DROP, 10-56
 - ECHO, 10-56
 - ELSE, 10-56, 10-57
 - END, 10-57
 - ERROR, 10-72
 - EXIT, 10-57, 10-58, 10-70
 - EXPOSE, 10-69
 - exposing variables, 10-69
 - EXTERNAL, 10-64, 10-65
 - FORM, 10-65
 - FUZZ, 10-65
 - HALT, 10-72
 - IF, 10-58, 10-59, 10-72
 - INTERPRET, 10-59, 10-72
 - IOERR, 10-72
 - ITERATE, 10-60
 - LEAVE, 10-60, 10-61
 - NOP, 10-61
 - NOVALUE, 10-72
 - NUMERIC, 10-61, 10-62, 10-65
 - OPTIONS, 10-62, 10-63
 - OTHERWISE, 10-63
 - PARSE, 10-64—10-68
 - PROCEDURE, 10-68, 10-69
 - PULL, 10-65, 10-69
 - RETURN, 10-70
 - SAY, 10-70
 - SELECT, 10-70—10-72
 - SHELL, 10-71
 - SIGL, 10-73
 - SIGNAL, 10-71—10-73
 - SOURCE, 10-65
 - SYNTAX, 10-72
 - UPPER, 10-64
 - VALUE, 10-65
 - VAR, 10-65
 - VERSION, 10-65
 - WHEN, 10-73
- AREXX introduction, 10-1—10-7
 - chapter organization, 10-3, 10-4
 - What do you have to know to use AREXX?, 10-3
 - What is AREXX?, 10-7
 - Who is AREXX for?, 10-2
- AREXX language features, 10-15, 10-16
- AREXX on the Amiga, 10-4—10-6

- displaying output, 10-13
- Interprocess communication and ports, 10-5, 10-6
- Multitasking, 10-5
- Multitasking and Interprocess communication together, 10-6
- AREXX Parsing and Templates, 10-66, 10-67, 10-146—10-154
 - additional notes, 10-154
 - multiple templates, 10-153
 - pattern parsing, 10-151, 10-152
 - positional markers, 10-152
 - scanning process, the, 10-149, 10-150
 - template objects, 10-148, 10-149
 - template structure, 10-146, 10-147
 - templates in action, 10-150, 10-151
- AREXX program examples, 10-16, 10-17
- AREXX support, ED, 9-25—9-27
- AREXX system files, 10-14, 10-15
 - SYS:LIBS Directory, 10-14
 - SYS:Rexxc Directory, 10-15
 - SYS:rexx, 10-15
 - SYSTEM: Directory, 10-14
- AREXX Tracing and Interrupts, 10-134—10-145
 - command inhibition, 10-139
 - display formatting, 10-136, 10-137
 - error processing, 10-141
 - external tracing flag, the, 10-142
 - failure level for commands, 10-142
 - global tracing console, 10-137, 10-138
 - interactive tracing, 10-140, 10-141
 - interrupts, 10-143—10-145
 - tracing options, 10-135, 10-136
 - tracing output, 10-138, 10-139
 - tracing prefix codes, 10-137
- AREXX, elements of, 10-26—10-50
 - boolean values, 10-38
 - clause classification, 10-34
 - clause continuation, 10-34
 - clauses, 10-31—10-33

- execution environment, 10-46
- expressions, 10-35
- external environment, 10-47
- format, 10-27
- input and output, 10-49
- internal environment, 10-47, 10-48
- numbers and numeric precision, 10-37—10-39
- operators, arithmetic, 10-39—10-44
- order of evaluation, 10-36, 10-37
- resource tracking, 10-49, 10-50
- stems and compound symbols, 10-44—10-46
- symbol resolution, 10-36
- tokens, 10-27—10-31
- ARG, 10-52, 10-64, 10-93
- arguments, EDIT, 9-68—9-72
- arrows, 5-18
- ASCII, 4-25, 4-29, 4-29
- ASK, 8-11
- ASSIGN, 8-12—8-16
- autopoint, 5-31
- AUTOPOINT, 8-16, 8-17
- AVAIL, 8-17, 8-18

B

- B2C, 10-94
- backing up your hard disk, 6-14—6-48, C-1, C-12
- backup disks, 1-27—1-37
- basic AmigaDOS commands, 7-11—7-28
 - CD, 7-12, 7-19—7-21, 7-25
 - COPY, 7-12, 7-18, 7-19, 7-23, 7-24, 7-50
 - DATE, 7-12, 7-27, 7-28
 - DELETE, 7-12, 7-24, 7-25
 - DIR, 7-12, 7-15—7-17
 - DISKCOPY, 7-12, 7-26, 7-27,
 - ENDSHELL, 7-12, 7-28, 7-41
 - FORMAT, 7-12, 7-25, 7-26

- INFO, 7-12, 7-15, 7-18
 - LIST, 7-12, 7-15, 7-17
 - MAKEDIR, 7-12, 7-18, 7-19
 - NEWSHELL, 7-12, 7-28
 - PATH, 7-12, 7-22, 7-23
 - RELABEL, 7-12, 7-26
 - RENAME, 7-12, 7-24, 7-49
 - SETCLOCK, 7-12, 7-27, 7-28
 - TYPE, 7-12, 7-24
 - Bindmonitor, 4-6, 4-10
 - BINDMONITOR, 8-19
 - BINDDRIVERS, 8-18
 - BITAND, 10-94
 - BITCHG, 10-94
 - BITCLR, 10-95
 - BITCOMP, 10-95
 - BITOR, 10-95
 - BITSET, 10-96
 - BITTST, 10-96
 - BITXOR, 10-96
 - blanker, 5-31, 5-32
 - BLANKER, 8-20, 8-21
 - boolean, C-3
 - boolean fields, C-5, C-6
 - boolean values, 10-38
 - bootable, 1-40
 - booting, 1-3
 - box gadget, 5-16
 - BREAK, 8-21, 8-22, 10-52
 - BREAK_C, 10-71
 - BREAK_D, 10-71
 - BREAK_E, 10-72
 - BREAK_F, 10-72
 - BRU argument reference section, C-20—C-35
 - control options, C-26—C-33
 - file selection options, C-33—C-35
 - modes, C-20—C-25
 - BRU command lines, C-7—C-9
 - BRU error messages, C-39—C-53
 - messages starting with filename, C-40—C-45
 - messages starting with warning, C-47—C-50
 - other messages, C-50—C-53
 - BRU with UNIX, C-36—C-39
 - control options, C-36—C-38
 - selection options, C-39
 - BRU, C-1
 - Brutab, C-2
 - Brutab fields, C-4, C-5
 - Built-in functions, 10-85, 10-87
- C
- C2B, 10-97
 - C2D, 10-97
 - C2X, 10-97
 - calculator, 5-4—5-6
 - CALCULATOR, 8-22
 - CALL, 10-53
 - cancelling, 2-19
 - CD, 7-12, 7-19—7-21, 7-25, 8-22, 8-23
 - CENTER or CENTRE, 10-98
 - center picture, 3-60
 - CHANGTASKPRI, 8-24
 - changing a Tool Type, 4-4, 4-5
 - changing the current directory, AmigaDOS, 7-19—7-21
 - changing the drive type, 6-80—6-86
 - changing the prompt, AmigaDOS, 7-34
 - changing the search path, AmigaDOS, 7-22, 7-23
 - check box, 2-40
 - Chip RAM, 2-26
 - circle gadget, 5-15
 - clause classification, 10-34
 - clause continuation, 10-34
 - clauses, 10-31—10-33
 - clear, 5-17
 - CLI, 4-6
 - click, 2-5

- clicking, 1-7
- Clip List, the, 10-89, 10-90
- Clock, 4-18—4-23
 - Alarm menu, 4-21
 - Date menu, 4-21
 - Mode menu, 4-20
 - Seconds menu, 4-20
 - Tool Types, 4-23
 - Type menu, 4-20
- CLOCK, 8-25, 8-26
- close gadget, 1-25, 2-37, 2-38
- CLOSE, 10-98
- CLOSEPORT, 10-130
- closing the Shell, AmigaDOS, 7-41
- CMD, 5-7, 5-8, 8-26, 8-27
- color correct, 3-57
- color selection gadget, 5-11, 5-13, 5-14
- color sliders, 5-11
- colors, 5-8—5-11
- COLORS, 8-27, 8-28
- command clauses, 10-74, 10-75
- command history, 7-35
- command inhibition, 10-82, 10-139
- command interface, the, 10-78, 10-79
- command line, AmigaDOS, 7-14
- command line characters, AmigaDOS, 7-29—7-31
- commands not in menus, MEmacs, 9-59—9-61
- commands, AmigaDOS, 7-2
- commands, EDIT, 9-68, 9-73—9-90
- Commodities, 4-27—4-29
- Commodities drawer, the, 5-27—5-30
- common additions to the startup files, 7-64—7-66
- COMPARE, 10-99
- components of the Brutab file, the, C-3—C-6
- COMPRESS, 10-98
- condition flags, AmigaDOS, 7-51, 7-52
- console window, 7-34
- continue gadget, 1-30—1-32, 1-36, 1-46, 1-47, 2-70, 2-71, 2-84
- continuous freehand gadget, 5-15
- control (Ctrl) key, 1-41
- control options, BRU, C-26—C-33
 - Amiga specific flags, C-26
 - asking BRU to wait for confirmation, C-32
 - exclude remotely mounted files, C-30
 - fast mode, C-28
 - interaction option, C-28
 - labeling an archive, C-29
 - pass over archive files by reading, C-29
 - pathname handling and expansion, C-30
 - set archive buffer size, C-26
 - setting the verbosity level, C-31, C-32
 - specify size of archive media, C-30
 - telling BRU to run without user intervention, C-26, C-27
 - turn on sparse file options, C-31
 - use paths as the archive file, C-27
 - use nbits for compression, C-29
 - use LZW file compression, C-33
- control options, BRU with UNIX, C-36—C-38
 - change the owner of extracted file, C-36
 - control string, C-36
 - do not reset access time, C-36
 - double buffer, C-37
 - filesystem, C-38
 - ignore unresolved links, C-37
 - interaction option, C-37
 - limit directory expansions to same mounted
 - select files owned by user, C-38
- COPIES, 10-99
- COPY, 7-12, 7-18, 7-19, 7-23, 7-24, 7-50, 8-28—8-30
- copying a disk, 2-68—2-74

- copying a disk in AmigaDOS, 7-26, 7-27
- copying a drawer, 2-72—2-74
- copying a project or tool, 2-72
- copying by dragging, 2-73, 2-74
- copying software to your hard disk, 6-4—6-6
- CPU, 8-31—8-33
- creating a new directory, AmigaDOS, 7-18, 7-19
- creating a new drawer, 2-59
- current directory, AmigaDOS, 7-19—7-21
- current line, EDIT, 9-65, 9-66
- cursor, 2-22
- customizing ED, 9-21—9-24
- customizing MEmacs, 9-61, 9-62
- customizing the window, AmigaDOS, 7-38—7-40
- cutting and pasting, AmigaDOS, 7-37, 7-38
- cycle gadget, 2-41
- cylinders, 1-31, 1-36, 6-2

D

- D2C, 10-99
- D2X, 10-100
- data, 1-43
- DATATYPE, 10-101, 10-102
- DATE, 7-12, 7-27, 7-28, 8-33, 8-34, 10-100, 10-101
- dead keys, 5-25
- DELETE, 7-12, 7-24, 7-25, 8-35, 8-36
- deleting a Tool Type, 4-4
- delimiters, ED, 9-5
- DELSTR, 10-102
- DELWORD, 10-102
- densities, B-1
- depth gadget, 1-17, 1-18, 2-31, 2-32
- destination disk, 1-29, 1-33
- device handlers, 7-85

- device name, AmigaDOS, 7-4
- devices, AmigaDOS, 7-3—7-5
- DEVS: directory, the, 7-80—7-85
 - keymaps, 7-84, 7-85
 - Mountlist, 7-81—7-84
 - printers, 7-85
- DF0:, 1-34
- DIGITS, 10-65, 10-102
- DIR, 7-12, 7-15—7-17, 8-36—8-38
- directories, AmigaDOS, 7-5, 7-6
- disk operating system, 7-1
- disk swap, 1-29, 1-33
- DISKCHANGE, 8-38, 8-39
- DiskCopy, 4-6, 4-11
- DISKCOPY, 1-33, 1-36, 7-12, 7-26, 7-27, 8-39, 8-40
- DISKDOCTOR, 8-40—8-42
- disks gadget, 2-24
- Display, 4-18, 4-19
- DISPLAY, 8-42—8-44
- display box, 2-43
- display formatting, 10-136, 10-137
- display mode, 3-3
- displaying output, AREXX, 10-13
- dithering, 3-60—3-62
- DO, 10-53—10-56
- double-click, 1-12, 2-11, 3-9
- drag selection, 2-9, 2-10
- dragging, 1-8, 1-35, 2-11—2-14
 - dragging a screen, 2-13, 2-14
 - dragging a window, 2-12, 2-13
 - dragging an icon, 2-11, 2-12
- DROP, 10-56
- drawer icon, 2-30
- drawers, 1-14, 1-15
- drive names, 1-28

E

- ECHO, 8-44, 8-45, 10-56
- ED, 8-45, 9-1—9-27

EDIT commands, 9-73—9-91
 changing command, input, and output files, 9-87—9-91
 editing the current line, 9-75—9-77
 editing line windows, 9-80—9-82
 ending EDIT, 9-91
 inserting and deleting lines, 9-77—9-79
 inspecting the source file, 9-85
 making global changes, 9-86, 9-87
 renumbering lines, 9-83, 9-84
 selecting the current line, 9-73—9-75
 splitting and joining lines, 9-82, 9-83
 verifying lines, 9-84
Edit menu, MEmacs, 9-38—9-43
EDIT, 8-45, 9-1, 9-63—9-91
editing, AmigaDOS, 7-35—7-37
editing a drive type or defining a new drive type, 6-82—6-86
editing the Startup sequence, 7-62—7-64
editor menus and presets drawer, 3-80—3-83
 Edit menu, the, 3-82
 Options menu, the, 3-82
 Project menu, the, 3-81
 Using the presets drawer, 3-83
editors, 9-1
ELSE, 8-46, 8-47, 10-56, 10-57
empty disk icon, 1-48
END, 10-57
ENDCLI, 8-47
ENDIF, 8-47
ENDSHELL, 7-12, 7-28, 7-41, 8-48
ENDSKIP, 8-48
environmental variables, AmigaDOS, 7-52—7-55
EOF, 10-103
error processing, 10-141
ERROR, 10-72

ERRORTTEXT, 10-103
EVAL, 8-49—8-52
Exchange, 4-18, 4-27, 4-28
EXCHANGE, 8-52
EXECUTE, 8-53—8-58
execution environment, 10-46
EXISTS, 10-103
EXIT, 10-57, 10-58, 10-70
EXPORT, 10-103, 10-104
EXPOSE, 10-69
exposing variables, 10-69
expressions, 10-35
extended commands, ED, 9-10—9-19
 altering text, 9-15, 9-16
 block control, 9-16, 9-17
 cursor control, 9-14, 9-15
 program control, 9-11—9-14
 searching and exchanging, 9-18, 9-19
extended selection, 2-11
EXTERNAL, 10-64, 10-65
External AREXX programs, 10-86
external environment, 10-47
External Function Libraries, 10-88, 10-89
external tracing flag, the, 10-142
Extras 2.0 disk, 1-36, 4-9
Extras menu, MEmacs, 9-54—9-59
extras programs, the, 5-1

F

FAILAT, 8-58, 8-59
failure level for commands, 10-142
fast RAM, 2-26
FAULT, 8-60
features of the Shell, 7-34—7-41
 closing the Shell, 7-41
 customizing the window, 7-38—7-40
 cutting and pasting, 7-37, 7-38
 editing, 7-35—7-37
file attributes, 2-65
file selection options, BRU, C-33—C-35

- selecting files by date, C-33—C-35
- unconditional file type extraction, C-35
- file system, AmigaDOS, 7-2, 7-3
- file system maintenance, hard drive, 6-90—6-95
- FILENOTE, 8-60, 8-61
- files, AmigaDOS, 7-6, 7-7, 7-23—7-25
- fill gadget, 5-17
- FIND, 10-104
- FixFonts, 4-6, 4-11, 4-12
- FIXFONTS, 8-62
- FKey, 5-33—5-35
- FKEY, 8-62, 8-63
- floppy disk, 1-2, 1-3
- FONT, 8-63, 8-64
- Font editor, the, 3-26—3-30
 - font gadget, 3-28, 3-29
 - text/field, 3-29, 3-30
 - text radio buttons, 3-27, 3-28
- FONTS: directory, the, 7-89
- Fonts drawer, 4-11, 4-12
- FORM, 10-65, 10-104
- Format, 4-6, 4-12, 10-27
- FORMAT, 7-12, 7-25, 7-26, 8-65
- formatting a disk, 1-44—1-47
- freehand gadget, 5-14
- FREEMEM, 10-129, 10-130
- FREESPACE, 10-104, 10-105
- Function Libraries and Function Hosts, 10-86
- FUZZ, 10-65, 10-105
- G**
- gadgets, 1-15
- GET, 8-66
- GETARG, 10-131
- GETCLIP, 10-105
- GETNV, 8-66, 8-67
- GETPKT, 10-131
- GETSPACE, 10-106

- getting information about disks, AmigaDOS, 7-15—7-18
- getting started, 1-2—1-5
- ghosted menu, 2-16
- ghosted menu items, 1-13
- global tracing console, 10-137, 10-138
- GraphicDump, 5-12
- GRAPHICDUMP, 8-67

H

- HALT, 10-72
- hard disk, 1-2
- hard disk partitions, 6-3, 6-4
- hard drive preparation, partitioning and formatting screen, 6-51—6-53
- HASH, 10-106
- HDBackup, 6-14—6-48, C-1
 - checking differences, 6-38—6-40
 - creating a full backup, 6-15—6-27
 - creating an incremental backup, 6-27—6-36
 - file selection gadget, 6-30
 - include and exclude gadgets, 6-29
 - selected files and selected size display, 6-29
 - selecting files by archive bit status, 6-30, 6-31
 - selecting files by date, 6-33, 6-34
 - selecting files by pattern, 6-31, 6-32
 - selecting files by size, 6-34, 6-35
 - smaller log file option, 6-35, 6-36
 - file compression option, 6-36—6-38
 - inspecting a backup, 6-40—6-42
 - restoring files, 6-42—6-44
 - tool types, 6-45—6-48
- HDDToolbox, 6-49—6-93
 - advanced options with partitioning, 6-60—6-62
 - adding bad blocks to the bad blocks list, 6-78—6-80

- changing the drive type, 6-80—6-85
- file system maintenance, 6-89—6-93
 - to add a new file, 6-91, 6-92
 - to delete a file system, 6-92
 - to update an existing file system, 6-93
- hard drive preparation, partitioning and formatting screen, 6-50—6-53
- low level formatting, 6-72—6-74
- locating bad blocks, 6-75—6-77
- modifying file systems, 6-86—6-89
- partitioning, 6-53—6-59
 - adding a new partition, 6-57
 - adjusting the size of a partition, 6-56
 - deleting a partition, 6-58
 - renaming a partition, 6-57
 - saving and formatting your new partitions, 6-59
 - sliding a partition within a partitioning bar, 6-56, 6-57
 - using HDTtoolbox's default setup for the drive, 6-58, 6-59
 - preparing a new hard disk, 6-63—6-72
- HI, 10-155
- Highlight Menu, 5-20—5-21
 - Backfill, 5-21
 - Compliment, 5-21
 - Image, 5-22
- host address, the, 10-75—10-78
- hot key, 5-29
- I
- IconEdit, 5-13—5-18
 - arrows, 5-18
 - box gadget, 5-15
 - circle gadget, 5-15
 - clear, 5-17
 - color selection gadget, 5-14
 - continuous freehand gadget, 5-15
 - fill gadget, 5-17
 - freehand gadget, 5-14
 - line gadget, 5-16
 - magnified view box, 5-14
 - Normal/Selected radio buttons, 5-17, 5-18
 - Undo, 5-17
- IconEdit menus, 5-19—5-23
 - Project, 5-19, 5-20
 - Type, 5-20
 - Highlight, 5-20, 5-21
 - Images, 5-22, 5-23
 - Misc, 5-23
- ICONEDIT, 8-68
- icons, 1-4, 2-49, 2-50
- Icons menu, the, 1-11, 2-67—2-87
 - Copy, 2-68—2-74
 - Delete . . . , 2-81, 2-82
 - Empty Trash, 2-86, 2-87
 - Format Disk . . . , 2-82—2-85
 - Information . . . , 2-76—2-79
 - Leave Out, 2-80
 - Open, 2-67, 2-68
 - Put Away, 2-81
 - Rename . . . , 2-75, 2-76
 - Snapshot, 2-79
 - Unsnapshot, 2-80
- ICONTROL, 8-68, 8-69
- IControl editor, the, 3-74—3-78
 - coercion, 3-76, 3-77
 - command keys, 3-75
 - mouse screen drag, 3-75, 3-76
 - screen menu snap, 3-77
 - text gadget filter, 3-77, 3-78
 - verify timeout, 3-74, 3-75
- ICONX, 8-69, 8-70
- IF, 8-70—8-72, 10-58, 10-59, 10-72
- IFF ILBM format, 4-23
- IHelp, 5-35, 5-36
- IHELP, 8-72, 8-73
- Images menu, 5-22, 5-23
 - Copy, 5-22
 - Exchange, 5-22

- Load, 5-22
- Restore, 5-23
- Save IFF Brush, 5-23
- immediate commands, ED, 9-6—9-9
 - changing case, 9-9
 - deleting text, 9-9
 - inserting text, 9-8
 - moving the cursor, 9-7, 9-8
- IMPORT, 10-106
- INDEX, 10-107
- INFO, 7-12, 7-15, 7-18, 8-73, 8-74
- .info files, AmigaDOS, 7-7
- initializing your printer, 5-24
- InitPrinter, 5-24
- INITPRINTER, 8-74
- INPUT, 8-75
- input and output, 10-49
- Input editor, 3-7
- INSERT, 10-107
- INSTALL, 8-76, 8-77
- instructions, AREXX, 10-18—10-24,
10-50—10-73
- interactive tracing, 10-140, 10-141
- interlaced, 3-31
- internal environment, 10-47, 10-48
- Internal functions, 10-85, 10-87
- INTERPRET, 10-59, 10-72
- Interprocess communication and ports,
10-5, 10-6
- interrupting BRU while it is operating,
C-9
- interrupts, 10-143—10-145
- introduction to AmigaDOS, 7-1, 7-2
- IOERR, 10-72
- IPREFS, 8-77
- ITERATE, 10-60

J

- JOIN, 8-77, 8-88

K

- key repeat delay, 3-10
- key repeat rate, 3-11
- key repeat test, 3-11
- keyboard shortcuts, 1-12
- keymap icon, 4-16
- keyshow, 5-24—5-26
- KEYSHOW, 8-78
- KEYWORD, 4-2

L

- L: directory, the, 7-85—7-89
 - Aux-Handler, 7-86
 - Pipe-Handler, 7-86, 7-87
 - Port-Handler, 7-87
 - Speak-Handler, 7-87—7-89
- LAB, 8-78
- language features, AREXX, 10-15, 10-16
- LASTPOS, 10-107, 10-108
- LEAVE, 10-60, 10-61
- LEFT, 10-108
- LENGTH, 10-108
- Library List, the, 10-83, 10-84
- LIBS: directory, the, 7-89, 7-90
- line gadget, 5-16
- Line menu, MEmacs, 9-48—9-50
- LINES, 10-108
- LIST, 7-12, 7-15, 7-17, 8-79—8-82
- LOADWB, 8-83
- locating bad blocks, 6-57—6-77
- LOCK, 8-83, 8-84
- low level formatting, 6-72—6-74

M

- magnified view box, 5-14
- MAKEDIR, 7-12, 7-18, 7-19, 8-84, 8-85
- MAKELINK, 8-85
- making backup copies of disks, 1-27—
1-37

- using one disk drive, 1-29—1-33
- using two disk drives, 1-33—1-37
- making commands resident,
 - AmigaDOS, 7-67, 7-68
- making room on your Workbench disk,
 - AmigaDOS, 7-69—7-71
- MAX, 10-109
- MEmacs, 7-47—7-49, 9-1, 9-28—9-62
- MEmacs screen, 5-9, 5-10
- menu bar, 1-9, 2-26
- menu button, 1-7, 1-9, 2-15
- menu commands, MEMacs, 9-32, 9-33
- menu items, 1-9
- MIN, 10-109
- Misc menu, 5-23
 - Auto Top Left, 5-23
 - Grid, 5-23
 - Remap B/W, 5-23
- modes, BRU, C-20—C-25
 - creating an archive, C-20
 - differences in size, C-20, C-21
 - estimating the size, C-21, C-22
 - extraction, C-25
 - give information on archive header, C-22
 - inspecting an archive, C-23
 - listing the table of contents, C-24
 - print the BRU help screen, C-23
- modifying file systems, hard drive, 6-87—6-90
- Monitors drawer, 4-7, 4-9, 4-10
- MonitorStore drawer, 4-7, 4-8, 5-3
- More, 4-18, 4-29—4-31
- MORE, 8-86
- MOUNT, 8-87
- mouse, 2-5—2-7
- mouse shortcut, 1-12
- mouse speed, 3-8—3-10
- mouse techniques, 2-5—2-7
- Move menu, MEMacs, 9-46, 9-47

- moving the Workbench screen, 2-27
- multiple commands, EDIT, 9-72, 9-73
- multiple templates, 10-153
- Multiscan monitor, 4-7
- multitasking, 1-15, 10-5

N

- naming AREXX programs, 10-13—10-15
- naming files, 1-54
- naming files and directories,
 - AmigaDOS, 7-9
- new drawer, 1-49, 1-50
- new features of ED, 9-2
- NEWCLI, 8-88
- NEWSHELL, 7-12, 7-28, 8-88—8-90
- NoCapsLock, 5-37
- NOCAPSLOCK, 8-90
- NoFastMem, 4-6, 4-12, 4-13
- NOFASTMEM, 8-91
- NOP, 10-61
- NTSC monitor, 4-7
- numbers and numeric precision, 10-37, 10-38
- NUMERIC, 10-61, 10-62

O

- OPEN, 10-109
- opening an icon, 2-67, 2-68
- opening/closing Shellwindows,
 - AmigaDOS, 7-28
- OPTIONS, 10-62, 10-63
- order of evaluation, 10-36
- organizing information on a disk, 1-47—1-51
- other Workbench directories, 7-78—7-90
 - DEVS: directory, 7-80—7-85
 - FONTS: directory, 7-89
 - L: directory, 7-85—7-89
 - LIBS: directory, 7-89, 7-90

- S: directory, 7-78—7-80
- OTHERWISE, 10-63
- output processing, EDIT, 9-67, 9-68
- OVERLAY, 10-110
- OVERSCAN, 8-91, 8-92
- Overscan editor, the, 3-42—3-48
 - edit standard overscan . . . , 3-46, 3-47
 - edit text overscan . . . , 3-44, 3-45
 - screen sizes, 3-47, 3-48

P

- PAL monitor, 4-7
- PALETTE, 8-92
- Palette editor, the, 3-12—3-14
- parent, 2-25
- PARSE, 10-64—10-68
- partitioning, 6-51—6-59
 - adding a new partition, 6-57
 - adjusting the size of a partition, 6-56
 - deleting a partition, 6-58
 - sliding a partition within a
 - partitioning bar, 6-56, 6-57
 - using HDToolbox's default setup for the drive, 6-58, 6-59
- partitions, 6-3, 6-4
- PATH, 7-12, 7-22, 7-23, 8-93, 8-94
- paths, 1-51—1-53, 7-8
- pattern matching, AmigaDOS, 7-31, 7-32
- pattern parsing, 10-151, 10-152
- peripheral devices, AmigaDOS, 7-4, 7-5
- point, 2-5
- pointer, 1-5, 2-5
- POINTER, 8-95
- Pointer editor, the, 3-22—3-25
- POS, 10-110
- positional markers, 10-152
- PRAGMA, 10-110—10-112
- preferences, 3-1
- Prefs drawer, the, 3-2, 3-3
- preparing a new hard disk, 6-63—6-72
- PRINTER, 8-96

- printer drivers, 3-3, B-1—B-21
 - CalComp_ColorMaster, B-2, B-3
 - CBM_MPS1000, B-3, B-4
 - Commodore MPS-1250 printer, B-6, B-7
 - Diablo_630, B-4
 - Epson Q, B-5
 - EpsonX, B-6, B-7
 - EpsonXOld, B-8
 - Howtek_Pixelmaster, B-9
 - HP_DeskJet, B-9, B-10
 - HP_LaserJet, B-10
 - HP_PaintJet, B-11
 - HP_ThinkJet, B-11
 - ImagewriterII, B-12
 - NEC_Pinwriter, B-13
 - Okidata_2931, B-14
 - Okidata_92, B-15
 - Okimate_20, B-15, B-16
 - Seiko_5300, B-16, B-17
 - Tektronix_4693D, B-17
 - Tektronix_4696, B-18
 - Toshiba_P351C, B-19
 - Toshiba_P351SX, B-20
 - Xerox_4020, B-21
- Printer editor, the, 3-49—3-56
 - left margin, 3-52
 - paper length, 3-52
 - paper size, 3-54
 - paper type, 3-53
 - print pitch, 3-55
 - print quality, 3-56
 - print spacing, 3-55
 - printer port, 3-53
 - right margin, 3-53
- printer escape sequences, B-22—B-26
- Printer Graphics editor, the, 3-57—3-69
 - aspect, 3-64
 - center picture, 3-60
 - color correct, 3-57
 - density, 3-69
 - dithering, 3-60—3-62
 - height, 3-69

- image, 3-64
- left offset/no. inches, 3-60
- limits/type, 3-66—3-68
- PrinterGFX icon, 3-57
- scaling, 3-63
- shade, 3-65
- smoothing, 3-59
- threshold, 3-65, 3-66
- tips for printing screen dumps, 3-58
- width, 3-69
- PRINTERGFX, 8-96, 8-97
- PrintFiles, 5-26, 5-27
- PRINTFILES, 8-97
- PROCEDURE, 10-68, 10-69
- program examples, AREXX, 10-16, 10-17
- Project menu, 5-19, 5-20
 - New, 5-19
 - Open . . . , 5-19
 - Save, 5-19
 - Save As . . . , 5-19
 - Save As Default Icon, 5-19
 - Quit, 5-20
- Project menu, MEMacs, 9-33—9-37
- PROMPT, 8-98, 8-99
- prompt, AmigaDOS, 7-13
- prompts, EDIT, 9-66, 9-67
- PROTECT, 8-99—8-101
- pseudo-icon, 2-64, 2-65
- PULL, 10-69

Q

- QUIT, 8-101, 8-102
- quarter1 window, 1-50

R

- radio button, 2-42, 5-17, 5-18
- RAM Disk, 1-5, 1-14, 1-40, 7-72—7-74
- RANDOM, 10-112
- RANDU, 10-112, 10-113
- read/write heads, 6-2, 6-3
- READCH, 10-113

- READLN, 10-113
- rebooting the Amiga, 1-40, 1-41
- recoverable RAM disk, the, 7-74—7-77
- redirection, AmigaDOS, 7-33, 7-34
- RELABEL, 7-12, 7-26, 8-102
- relative path, C-17
- REMLIB, 10-113, 10-114
- REMRAD, 8-103
- RENAME, 7-12, 7-24, 7-49, 8-103, 8-104
- rename menu item, 1-38
- renaming an icon, 2-75, 2-76
- renaming your backup disks, 1-38—1-40
- repeating commands, ED, 9-20
- Reports disk window, 1-49
- requesters, 1-25, 2-20—2-25
 - action requesters, 2-21
 - file requester, 2-24, 2-25
 - text requester, 2-22, 2-23
- RESIDENT, 8-104—8-107
- resolution, 3-30
- resource tracking, 10-49, 10-50
- restoring files to your hard disk, C-17—C-19
- RETURN, 10-70
- return code, 7-51
- REVERSE, 10-114
- review and additional notes, AREXX,
10-24—10-26
- REXXC Directory, command utilities,
10-154—10-157
 - HI, 10-155
 - RX, 10-155
 - RXC, 10-156
 - RXSET, 10-155
 - TCC, 10-156
 - TCO, 10-156
 - TE, 10-156, 10-157
 - TS, 10-156
 - WaitForPort, 10-157
- RexxMast, 4-6
- REXXSUPPORT.LIBRARY Functions,
10-129—10-134
 - ALLOCMEM, 10-129, 10-130

- CLOSEPORT, 10-130
- FREEMEM, 10-129, 10-130
- GETARG, 10-131
- GETPKT, 10-131
- OPENPORT, 10-130, 10-132
- REPLY, 10-130, 10-132
- SHOWDIR, 10-132, 10-133
- SHOWLIST, 10-133
- STATEF, 10-133
- WAITPKT, 10-131, 10-134
- RIGHT, 10-114
- ROM, 3-26, 4-10, 4-11
- root block, 2-85
- RUN, 8-107, 8-108
- running programs, AmigaDOS, 7-46—7-51
- RX, 10-155
- RXC, 10-156
- RXSET, 10-155
- S
- S: directory, the, 7-78—7-80
 - ED-Startup, 7-79
 - HDBackup.config, 7-79
 - PCD, 7-80
 - SPat, DPat, 7-79
- sample path chart, 1-53
- Say, 4-18, 4-32—4-35
- SAY, 8-108, 8-109, 10-70
- scaling, 3-63
- scanning process, the, 10-149, 10-150
- SCREENMODE, 8-109, 8-110
- ScreenMode editor, the, 3-37—3-41
 - AutoScroll, 3-41
 - choose display mode, 3-37
 - colors, 3-41
 - height, 3-40
 - properties of the selected mode, 3-38
 - screen sizes, 3-39
 - width, 3-40
- script commands, AmigaDOS, 7-50, 7-51
- scripts, AmigaDOS, 7-2, 7-49—7-51
- scroll arrows, 1-24, 2-24, 2-35
- scroll bars, 1-21—1-23, 2-24, 2-34
- scroll boxes, 1-21, 2-34
- scroll gadgets, 1-21—1-24, 2-33—2-35, 2-43, 2-44
- scrolling, 1-15
- scrolling list, 2-43
- SEARCH, 8-110, 8-111
- Search menu, MEmacs, 9-52—9-54
- search path, 7-21—7-23
- sectors, 6-2
- SEEK, 10-114
- SELECT, 10-70, 10-71
- selecting, 2-8, 2-9
- selecting multiple icons, 2-9—2-11
- selection button, 1-7, 1-8, 2-8—2-11
- selection gadget, 2-45
- selection options, BRU with UNIX, C-39
- SERIAL, 8-112
- Serial editor, the, 3-70—3-73
 - baud rate, 3-70, 3-71
 - bits/char, 3-73
 - handshaking, 3-71, 3-72
 - input buffer size, 3-71
 - parity, 3-72
 - stop bits, 3-73
- SET, 8-113
- SETCLIP, 10-115
- SETCLOCK, 7-12, 7-27, 7-28, 8-114
- SETDATE, 8-115
- SETENV, 8-116
- SETFONT, 8-117
- SetMap, 4-7, 4-13—4-17
- SETMAP, 8-118
- SETPATCH, 8-119
- setting AREXX to start automatically, 10-9—10-12
- setting environment variables for BRU, C-6, C-7
- setting the clock, AmigaDOS, 7-27, 7-28
- Shell, the, 7-13, 7-14
- SHELL, 10-71
- Shell program, 2-4
- Shell-startup file, the, 7-41—7-46

- changing the prompt, 7-43
- using aliases, 7-41, 7-42
- using escape sequences, 7-43—7-46
- SHOW, 10-115
- SHOWLIST, 10-133
- SIGN, 10-115
- SIGNAL, 10-71—10-73
- single floppy disk systems, AmigaDOS,
 - 7-66, 7-67
- sizing gadget, 1-20, 2-35, 2-36
- SKIP, 8-119, 8-120
- slider bar, 2-46
- slider box, 2-46
- slider gadget, 2-46
- slider value, 2-46
- smoothing, 3-59
- SORT, 8-121
- source disk, 1-29, 1-33
- SOURCELINE, 10-116
- SPACE, 10-116
- special AmigaDOS characters, 7-29—
 - 7-34
 - command line characters, 7-29—7-31
 - pattern matching, 7-31, 7-32
 - redirection, 7-33, 7-34
- STACK, 8-122
- standard escape sequences for console
 - windows, AmigaDOS, 7-45
- starting AREXX on the Amiga, 10-7—
 - 10-12
 - automatically, 10-8
 - manually, 10-8
 - setting AREXX to start automatically,
 - 10-9—10-12
 - starting AREXX through the shell,
 - 10-8
- starting ED, 9-3, 9-4
- starting EDIT, 9-64, 9-65
- starting MEmacs, 9-28, 9-29
- Startup-sequence, the, 7-55—7-62
- STATEF, 10-133
- STATUS, 8-123
- stems and compound symbols, 10-44—

- 10-46
- STORAGE, 10-117
- STRIP, 10-117, 10-118
- submenus, 2-17, 2-18
- SUBSTR, 10-118
- SUBWORD, 10-119
- swap requester, 1-30
- SYMBOL, 10-118, 10-119
- symbol resolution, 10-36
- syntax and search order, 10-85—10-89
- SYS:LIBS Directory, 10-14
- SYS:rexx, 10-15
- SYS:Rexxc Directory, 10-15
- SYSTEM: Directory, 10-14
- System drawer, the, 4-5—4-16
 - AddMonitor, 4-6—4-10
 - BindMonitor, 4-6, 4-10
 - CLI, 4-6
 - DiskCopy, 4-6, 4-11
 - FixFonts, 4-6, 4-11, 4-12
 - Format, 4-6, 4-12
 - NoFastMem, 4-6, 4-12, 4-13
 - RexxMast, 4-6
 - SetMap, 4-7, 4-13—4-17

T

- TCC, 10-156
- TCO, 10-156
- TE, 10-156
- template objects, 10-148, 10-149
- template structure, 10-146, 10-147
- templates in action, 10-150, 10-151
- text gadget, 1-39, 2-22, 2-23, 2-47, 2-48
- text string, ED, 9-5
- TIME, 8-124, 10-119
- Time editor, 3-5, 3-6
- tips for printing screen dumps, 3-58
- title bar, 1-16, 1-17, 2-26, 2-30
- Tool Types, 2-79, 4-1—4-5
- Tools drawer, the, 5-3, 5-4
- Tools menu, the, 2-87
- Topaz, 4-11
- TRACE, 10-120

- tracing options, 10-135, 10-136
- tracing output, 10-138, 10-139
- tracks, 6-1, 6-2
- TRANSLATE, 10-120
- trashcan, 2-86, 2-87
- TRIM, 10-121
- troubleshooting, hard disk, 6-7, 6-8
- TRUNC, 10-121
- TS, 10-156
- tutorial, 1-1
- TYPE, 7-12, 7-24, 8-124, 8-125
- Type menu, 5-20
 - disk, 5-20
 - drawer, 5-20
 - garbage, 5-20
 - project, 5-20
 - tool, 5-20
- types of commands, AmigaDOS, 7-11—7-13
- types of displays, 3-30—3-36
 - A2024, 3-36
 - Hires, 3-33
 - possible display modes, 3-32
 - Productivity mode, 3-35
 - SuperHires, 3-34

U

- UNALIAS, 8-125
- unconditional file type extraction, BRU
 - with UNIX, C-39
- undo, 5-17
- UNSET, 8-125
- UNSETNV, 8-126
- UPPER, 10-64, 10-121
- using a hard disk, 6-1
- using aliases, AmigaDOS, 7-41, 7-42
- using AmigaDOS, 7-1
- using application software, 1-42—1-54
 - formatting a disk, 1-44—1-47
 - naming files, 1-54
 - organizing information on a disk, 1-47—1-51

- paths, 1-51—1-53
- using AREXX with command shells, 10-81
- using ASSIGN'S PATH option, 7-68, 7-69
- using BRU—a tutorial, C-10—C-19
 - adding control and selection options, C-14—C-17
 - backing up only a few files or directories, C-13
 - backing up your entire hard disk, C-12
 - combining modes, C-14
 - current directory, C-10—C-12
 - estimating the size and creating a backup of the current directory, C-10—C-12
 - restoring files to your hard disk, C-17—C-19
- using colors, 5-10, 5-11
- using commands in macro programs, 10-79, 10-80
- using ED, 9-5—9-27
- using EDIT, 9-65—9-91
- using escape sequences, AmigaDOS, 7-43—7-46
- using HDBackup for the first time, 6-15
- using MEMacs, 9-29—9-62
- using menus, 1-11, 1-13, 2-15—2-18
- using the Amiga without a mouse, 2-6
- using the mouse, 1-6—1-10
 - menu button, 1-7, 1-9, 1-10
 - selection button, 1-7, 1-8
- Utilities drawer, the, 4-18—4-35
 - Clock, 4-18—4-23
 - Display, 4-1, 4-23—4-26
 - Exchange, 4-18, 4-27, 4-28
 - More, 4-18, 4-29—4-31
 - Say, 4-18, 4-32—4-35

V

- VALUE, 10-122
- VERIFY, 10-122

VERSION, 8-126, 8-127
volume name, 1-28

W

WAIT, 8-127
WaitForPort, 10-157
WAITPKT, 10-134
WBCONFIG, 8-128
WBPATTERN, 8-129, 8-130
WBStartup drawer, the, 4-35, 4-36
WHEN, 10-73
WHICH, 8-130, 8-131
WHY, 8-131, 8-132
window, 1-4
Window menu, the, 2-58—2-66
 Clean Up, 2-63
 Close, 2-61
 New Drawer, 2-59
 Open Parent, 2-60
 Select Contents, 2-62
 Show, 2-64, 2-65
 Snapshot, 2-63, 2-64
 Update, 2-61, 2-62
 View By, 2-65, 2-66
Window menu, MEMacs, 9-44, 9-45
WORD, 10-122
Word menu, MEMacs, 9-50, 9-51
WORDINDEX, 10-123
WORDLENGTH, 10-123
WORDS, 10-123
Workbench, 1-1
Workbench 2.0 disk, 1-16, 1-38, 1-40,
 2-4, 4-9
Workbench 2.0 disk icon, 1-2, 1-8, 1-11,
 1-34, 1-35
Workbench 2.0 disk window, 1-14
Workbench Configuration editor, the,
 3-79, 3-80
Workbench icon, 2-2, 2-3
Workbench menu, the, 2-2, 2-3, 2-50—
 2-57
 Backdrop, 2-51
 Execute Command . . . , 2-52—2-54

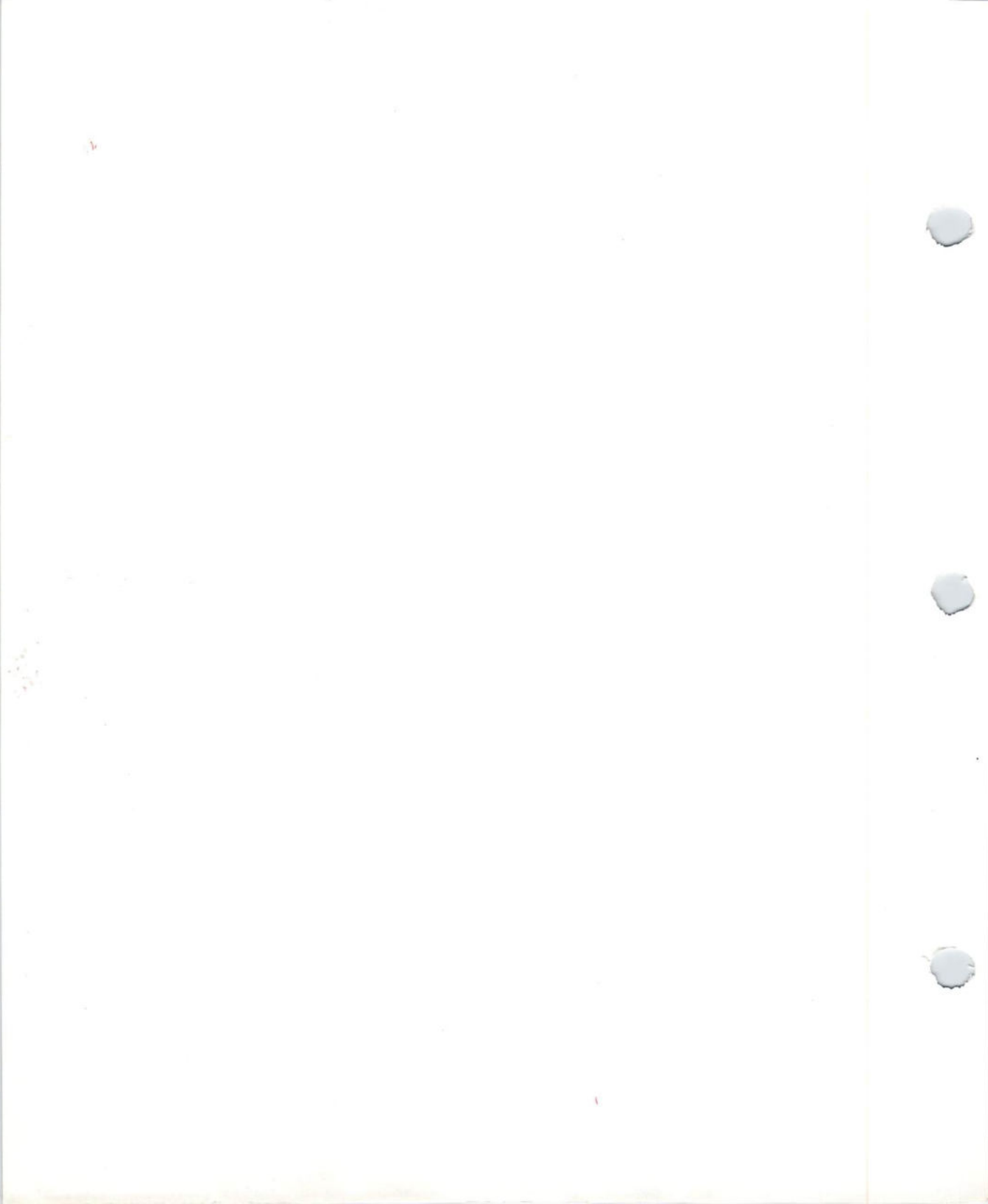
Keyboard Shortcuts, 2-51
Last Message, 2-55
Quit . . . , 2-56, 2-57
Redraw All, 2-54
Update All, 2-54
version, 2-55
Workbench Pattern editor, 3-15—3-22
 actual-size box, 3-20
 clear, 3-21
 color selection gadget, 3-19
 magnified view box, 3-20
 pattern, 3-20, 3-21
 presets gadget, 3-20
 test, 3-21
 undo, 3-21
Workbench programs, 4-1
Workbench screen, 1-4, 2-26—2-87
Workbench screen features, 2-2—2-4
Workbench system, 2-2—2-4
Workbench windows, 2-2, 2-3, 2-27—
 2-29
working with disks, AmigaDOS, 7-25—
 7-27
working with files, Amigados, 7-23—
 7-25
working with windows, 1-14—1-25
 close gadget, 1-25
 depth gadget, 1-17, 1-18
 scroll gadgets, 1-21—1-24
 sizing gadget, 1-20
 title bar, 1-16, 1-17
 zoom gadget, 1-19
write-enable, 1-27
write-protected, 1-43
WRITECH, 10-123
WRITELN, 10-124

X

X2C, 10-124
X2D, 10-124

Z

zoom gadget, 1-19, 2-31



 **Commodore®**
AMIGA®

Part No. 318344-02