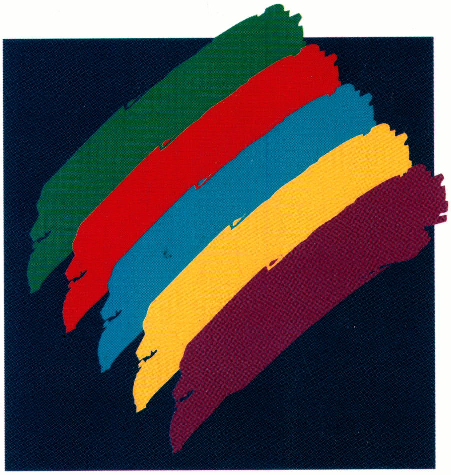
User's Guide

AmigaDOS

AMIGA



Commodore

User's Guide

AmigaDOS

AMIGA

Commodore

Printed in Hong Kong.

This book was produced using a variety of Commodore systems by Barbara Siwirski and Harry McCabe.

P/N: 368763-01

Table of Contents

Chapter 6
Editors

Welcome

The Commodore® Amiga® line of personal computers offers a unique combination of versatility, computing power, and usability.

The Amiga's Workbench™ Graphical User Interface (GUI) allows you to control the Amiga by using a mouse to select graphic symbols in the form of icons, or small pictures, and list-like menus. The Workbench is easy to learn and use because these icons and menus are displayed automatically. You do not have to memorize a long list of commands to accomplish a task.

Multitasking is the ability of the Amiga to run several independent programs at one time. Because the Amiga was designed from the start as a multitasking system with a GUI, it is fast and easy to switch between tasks (programs) when you need to. No only can programs run at the same time on the Amiga, but they can also share information and computer resources, allowing you to do more work without requiring additional software and memory.

The Workbench and multitasking are features common to all Amigas, from the single floppy models to the most advanced workstation-level machines that are used for three-dimensional graphics, multimedia, and video production.

This manual describes the AmigaDOS™ software, its components, and how to use it.

Using this Manual

AmigaDOS is the operating system software that controls the Amiga's functions. This document tells you how to use AmigaDOS. The following is a brief description of each chapter and appendix.

Chapter 1. Introducing AmigaDOS: This chapter defines the terms and concepts behind AmigaDOS.

Chapter 2. AmigaDOS Tutorial: This chapter briefly discusses the Shell interface and explains some of the most basic AmigaDOS commands via a step-by-step tutorial.

Chapter 3. Using the Shell: This chapter explains how to edit command lines, perform copy and paste operations, use escape sequences, and run programs.

Chapter 4. Working with Scripts: This chapter explains how to use script files to simplify repetitive and complex tasks.

Chapter 5. AmigaDOS Reference: This chapter lists and describes all the AmigaDOS commands.

Chapter 6. Editors: This chapter explains the three editors included with the Amiga: ED, MEmacs, and EDIT.

Appendix A. Error Messages: This chapter contains a list of possible problems and suggested solutions.

Appendix B. Additional Workbench Directories: This chapter describes the S:, DEVS:, L:, FONTS:, and LIBS: directories.

Appendix C. Single Floppy Disk Systems: This chapter tells you how to make the most of your system if you only have one floppy drive and no hard drive.

Documentation Conventions

The following conventions are used in this manual:

COMMANDS	Commands are displayed in all uppercase letters, however, they do not need to be entered this way. The Amiga ignores case differences in commands and filenames.
<n>	Angle brackets enclose variable information that you must supply. In place of <n>, substitute the value, text, or option desired. Do not enter the angle brackets when entering the variable.
`Courier`	Text appearing in the Courier font represents information displayed on your screen.
Key1 + Key2	Key combinations displayed with a + (plus) sign connecting them indicate pressing the keys simultaneously. For example, to exit the editor MEmacs, hold down the Ctrl key, and while holding it down, press C.
Key1, Key2	Key combinations displayed with a comma separating them indicate pressing the keys in sequence. For example, to open a file in ED, press the Esc key, followed by the O key, then followed by the P key.
1.	Numbered steps appearing in boldface represent information that is to be entered by the user.

Related Documentation

Amiga Workbench User's Guide

ARexx User's Guide

ROM Kernal manuals published by Addison-Wesley

Chapter 1

Introducing 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 that programs use and produce.
- handling storage and retrieval of information from floppy and hard disks.
- running 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.

This chapter introduces you to the basic concepts of AmigaDOS, such as:

- the hierarchical file system.
- the Shell interface.
- some basic AmigaDOS commands.

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 for performing repetitive tasks, to monitor the use of memory, and to perform other tasks unavailable through the Workbench.

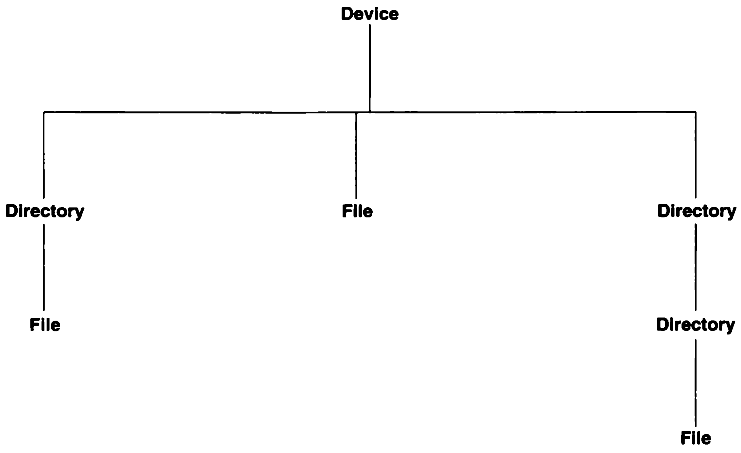
The commands are entered through a special window, known as a Shell window. You can start a Shell by opening the Shell icon in the

Workbench System drawer or by entering the NEWSHELL command. NEWSHELL will also open a Shell from within a Shell.

The next section explains how AmigaDOS stores information and how to gain access to the files and devices used by AmigaDOS.

The File System

AmigaDOS stores information on devices that are organized according to a system of directories, subdirectories, and files, known as a file system. Directories and files are arranged in a hierarchical system often referred to as a tree, since its branching diagram 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 like leaves.



Devices

You can store information on floppy disks, hard disks, magnetic tape or Ram disk. 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 Workbench: 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.

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 example, DF0: is the device name of the Amiga's internal floppy 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.

There is also RAD:, which is only lost if the system is powered off (but not when rebooted). Refer to Appendix C: "*Single Floppy Disk Systems*" for detailed information.

Device and volume names must be followed by a colon (:).

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, for example, copying a file to a printer or sending information through a modem. The standard device names are:

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.

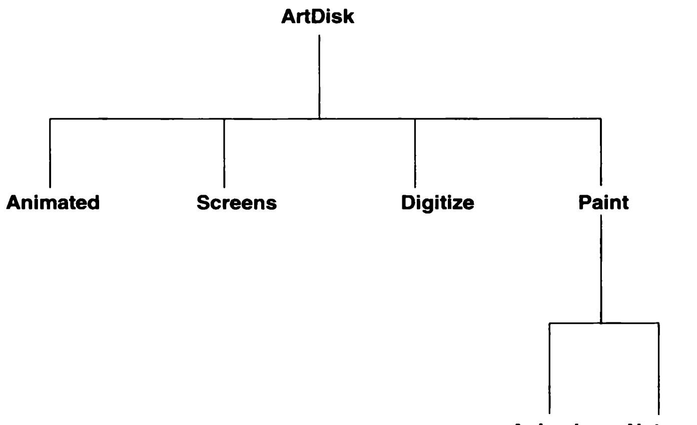
Directories

Directories allow you to group and classify related files, 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. Directories are the AmigaDOS version of the Workbench drawers.

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.)

Directories can contain files as well as other directories, called subdirectories.

For example, if you had a disk named ArtDisk and wanted to store animated files, screen shots, digitized art, and paint files, you could create four directories: Animated, Screens, Digitize, and Paint. If you had paint files that represented animals and nature scenes, you could create two subdirectories within the Paint directory: Animals and Nature. Your disk organization would look like this:



The Amiga supports an arbitrary number of nested directories. With many programs, the exact number is tied to the size of the stack used when invoking the programs. If you are trying to access a deeply nested directory from the Shell and the procedure fails, try to increase the size of the stack using the STACK command and try again.

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. Project icons represent data files. Data files contain the information created or used by a program, such as text, graphic, and spreadsheet files.

Another type of file used by the Amiga is a .info file. The .info files contain the icons that appear on the Workbench screen. Every file or directory that has an icon also has a corresponding .info file. For example, the Workbench disk contains two files for the Clock program: one called Clock which contains the Clock program, and one 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 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 example, if you use the COPY command to copy the Clock file from the Utilities directory to the System directory, 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.)

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

When you identify a file (for example, to copy it or rename it), you must specify the complete path that leads to it. The path includes:

- The volume or device name. A volume name refers to a specific disk, such as Workbench:. 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 lead to the file.
- The filename.

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.

For example, the path to the file Readme in subdirectory Animated, directory Demos, on the disk called MyDisk in drive DF0: would be:

```
DF0:Demos/Animated/Readme
```

or

```
MyDisk:Demos/Animated/Readme
```

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 (%).

Any capitalization you use in a filename will be preserved, but AmigaDOS is case insensitive. This means that it will recognize the name by the characters, not by the case — TextFile will be treated the same as textfile.

Although you can insert spaces in names, you should avoid them when working through AmigaDOS. If you do use names with spaces, you must enclose the entire path containing the name in double quotes. For example:

```
"DF0:Text File"
```

It is generally better to use an underscore (_) as a separator.

Caution **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.**

A keyword is a special word recognized by an AmigaDOS command. Many AmigaDOS commands use keywords to identify an argument or to specify an option. 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 example, if you have a directory named Files and you want to display information about all of its files and subdirectories, you would use the LIST Files command. But, this would conflict with the LIST FILES command which lists only the files in a directory (excluding subdirectories). To get around this, you would have to enter:

```
LIST "Files"
```

You can have more than one file with the same name as long as each file is in a different directory.

Quick Review

- A device refers to a physical device, like a disk drive or printer, or to a software device, like the Ram Disk or a window.
- A volume is another term for a hard disk partition or floppy disk. Volume names can usually be changed.
- You can refer to disks by volume name, such as MyDisk:, or by a device name, such as DF0:. The names 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.
- Files are named collections of data.
- A path is the series of device, directory and subdirectory names used to reach a particular file.

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 (Workbench:), 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 enter 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 directories 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 (*)

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 all subsequent text you type in 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 apostrophes 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:

Note The back apostrophe is found next to the 1.

Caution 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 example, 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, all with one command.

Several different wildcards allow you to specify the type of match. To remove the special effect of these characters and search for a wildcard character, preface the character with an apostrophe ('). For example, '?, will match ?, and '' (two single apostrophes) 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:

?	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.
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(B|C)D	Matches ABD or ACD.
~ (XYZ)	Matches anything but XYZ.
~ (#?XYZ)	Matches anything not ending in XYZ.
A#(BC)	Matches any name beginning with A followed by any number of BC combinations, such as ABC, ABCBC, and ABCBCBC.
A(B|D|%)#C	Matches ABC, ADC, AC (% is the null string), ABCC, ADCC, ACCC, etc..
[A-D]#?	Matches any name beginning with A, B, C or D.

The most frequently used combination is #? (matches any characters). Note that #? is equivalent to the * wildcard used by some other computer systems. For example, to delete all the .info files in the Picture directory, you would enter:

```
1> DELETE Picture/#?.info
```

Caution Be careful when using #?. You could accidentally delete the contents of a disk.

Redirecting Input and Output

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 example, console output usually goes to the current Shell window. If you enter:

```
1> DIR >Testfile DF0:
```

a list of the contents of the disk in DF0: will be sent to a file called Testfile. The output will not be shown on the screen. 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 argument for the DATE command. The command responds as if the contents of Datefile were entered at the keyboard.

Console output of a command only is redirected, not the data. For example:

```
1> COPY >Log Picdir to PicsArchive: ALL
```

still copies all the files in the Picdir directory to the PicsArchive disk and sends a list of the copied filenames to the Log file. The redirection argument appears immediately after the command and before the other arguments.

Redirect output and append material to an existing file using 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.

Chapter 2
AmigaDOS Tutorial

This section introduces the Shell and some of the most basic AmigaDOS commands. It does not give full instructions on every command, but it will familiarize you 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, it 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 (Read Only Memory). When an internal command is invoked, the system can access it immediately.

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.

The commands included in this section and their Workbench counterparts, if any, are shown in Table 2-1.

Table 2-1. Basic AmigaDOS Commands

Command	Function	Workbench Counterpart
DIR	Show files on disk	Menu Item Show All Files
LIST	Show files, with sizes, etc.	Menu Item View By Name
INFO	Show information on all disks	Title bars
MAKEDIR	Make a new directory	Menu Item New Drawer
COPY	Copy a file, directory, or disk	Menu Item Copy
CD	Change the current directory	Select another window
PATH	Add a directory to the search path	
TYPE	Display the contents of a file	Program More
RENAME	Rename a file or directory	Menu Item Rename
DELETE	Delete a file, directory, or disk	Menu Item Delete
FORMAT	Format a disk	Menu Item Format Disk
RELABEL	Rename a disk	Menu Item Rename
DISKCOPY	Copy a disk	Menu Item Copy
DATE	Set the correct date and time	Program Time editor
SETCLOCK	Save the date and time	Program Time editor
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, reboot your machine with the Workbench 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, although the commands work in the same way.

Caution If you have a single floppy disk system, please use a copy of the Workbench disk in order to avoid changing the original Workbench disk.

The Shell

You communicate with AmigaDOS through a Shell, a special window which accepts text input. A Shell is a variation of a CLI, a Command Line Interface. 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 Chapter 3. *"Using the Shell"*).

You enter AmigaDOS commands at a text prompt, usually ending in a > symbol. After entering 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 entered after the prompt is known as the command line.

The following is a tutorial of 25 numbered steps introducing some basic AmigaDOS commands. If you are not familiar with AmigaDOS, please perform each step from beginning to end. If you are familiar with AmigaDOS, you may skip to the command you would like to reference, keeping in mind that these are sequential steps, and were meant to be completed in their entirety. The entire tutorial should take approximately 45 minutes.

1. **Open a Shell window by opening the Shell icon in the System drawer.**

   When you see a prompt, such as 1.SYS:>, you are ready to begin entering commands. The Shell prompt may vary depending on your system. In the tutorial, 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. 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.

Using DIR

The DIR command generates a list of all the files and directories included on a disk or within a directory.

2. **At the Shell prompt, enter:**

```
1> DIR
```

You should see output similar to the following:

```
1> DIR
     Expansion (dir)
     System (dir)
     Rexxc (dir)
     Utilities (dir)
     L (dir)
     S (dir)
     Trashcan (dir)
     T (dir)
     C (dir)
     Devs (dir)
     Prefs (dir)
     WBStartup (dir)
     Libs (dir)
Devs.info                     Disk.info
Expansion.info                Prefs.info
System.info                   Trashcan.info
Utilities.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 Workbench disk window. The names of normal files are listed in two columns, sorted alphabetically. If you choose the Show All Files menu item after returning to the Workbench, drawer icons for all the other directories will appear in the disk window.

You can see a list of files included in a directory by entering DIR followed by the directory name.

3. **For example, enter:**

```
1> DIR Devs
```

You should see output similar to the following.

```
1> DIR Devs
    DOSDrivers (dir)
    Keymaps (dir)
    Monitors (dir)
    Printers (dir)
clipboard.device              DOSDrivers.info
Keymaps.info                  mfm.device
Monitors.info                 MountList
parallel.device               postscript_init.ps
printer.device                Printers.info
serial.device                 system-configuration
```

DIR generated a list of the files, and any subdirectories, stored in the Devs directory on the Workbench: 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 Workbench:.

Using LIST

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 the files, the protection bits for both files and directories, and the date and time the files or directories were created. The names are displayed in only one column, but unlike the DIR command, they are not sorted alphabetically. The possible protection bits for a file or directory are listed below:

s	the file is a script
p	the file is pure and can be made resident
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 5.

4. **To list the contents of the Workbench: disk, enter:**

```
1> LIST
```

A shortened version of the typical output is shown below.

```
1> LIST
Utilities      Dir   ----rwed  20-Jun-91   17:22:47
L              Dir   ----rwed  20-Jun-91   16:03:05
S              Dir   ----rwed  20-Jun-91   17:22:47
Devs.info      632   ----rw-d  20-Jun-91   15:25:33
Expansion.info 632   ----rw-d  20-Jun-91   16:00:00
Prefs.info     724   ----rw-d  20-Jun-91   14:22:53
System.info    632   ----rw-d  20-Jun-91   17:30:22
Utilities.info 632   ----rw-d  20-Jun-91   09:30:45
8 files - 13 directories - 37 blocks used
```

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. To list the contents of the Utilities directory, enter:

```
1> LIST Utilities
```

A shortened version of the output is shown below.

```
1> LIST Utilities
Directory "Utilities" on Tuesday 09-Jul-91
Clock              13876  ----rwed 20-Jun-91  8:25:44
Clock.info           557  ----rw-d 20-Jun-91  8:25:44
Display            24052  ----rwed 12-Jul-91  6:35:12
Display.info         786  ----rw-d 12-Jul-91  6:02:57
More               11936  --p-rwed 20-Jun-91  7:21:46
More.info            454  ----rw-d 12-Jul-91  6:02:59
6 files - 110 blocks used
```

You can also use LIST to get information about a specific file. For example:

6. To list the contents of the Fountain file located in the System directory, enter:

```
1> LIST System/Fountain
```

You should see output similar to the following:

```
1> LIST System/Fountain
Directory "System" on Tuesday 09-Jun-92
Fountain      68880 ----rwed  04-Jan-92   6:02:01
1 file - 137 blocks used
```

Using INFO

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 how much space is available 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. **At the Shell prompt, enter:**

   ```
   1> INFO
   ```

   You should see output similar to the following:

   ```
   1> INFO
   Mounted Disks:
   UnitSize Used Free  Full  Errs   Status     Name
   RAM: 11K   11    0  100%   0   Read/Write  Ram Disk
   DF0:879K 1723   35   98%   0   Read Only   Workbench2.1

   Volumes Available:
   Ram Disk [Mounted]
   Workbench2.1[Mounted]
   ```

Creating A New Directory

Using MAKEDIR

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 Workbench disk, enter:**

   ```
   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 after returning to the Workbench.

Using COPY

To create an icon for Testdir, you can make a copy of the .info file for any existing drawer icon. For example, 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. **Enter:**

```
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 Workbench 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 entering 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 first look for information, 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:.

Using CD

To make another directory the current directory, use the CD command.

10. To make Testdir the current directory, enter:

```
1> CD Testdir
```

Now, if you enter the DIR or LIST command, the output will show the contents of the Testdir directory. When you want to work with the files in Testdir, you simply have to enter the filename instead of the complete path.

The current directory is part of the standard Shell prompt, such as:

```
1.Workbench:Testdir>
```

Entering CD alone will display the name of the current directory. This is a way to verify the name of the current directory before you issue any commands, particularly if the standard prompt style has been changed.

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 change to a directory by entering 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 change to that directory.

When entering the directory name, be sure to specify it relative to the existing current directory. For example, if the current directory is the root directory, Workbench:, and you enter:

```
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 return to the root directory by entering:

```
1> Workbench
```

AmigaDOS will search forward in the directory structure for a directory called Workbench. When it does not find one, it will respond:

```
Workbench:  Unknown command
```

To change to a higher-level directory, enter the appropriate number of slashes or the full path. For example, to go back to the root directory, enter:

```
1> /
```

or

```
1> :
```

If the current directory is Work:Program/Documents/Letters, entering:

```
1> //
```

will move you back to the Work:Program directory. You could also enter:

```
1> Work:Program
```

Each slash is equal to one directory level.

Changing the Search Path

Just as changing the current directory can save you from having to enter 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 automatically looks in these directories, in sequence, to locate a program to execute. If the file is not in any of those directories,

```
Object Not Found
```

is displayed.

You can add any directory to the search path using the PATH command. Adding a directory to the search path eliminates the need to supply a path to the program each time you want to execute it. You simply need to enter the filename, and AmigaDOS will find it as if you had entered the complete path.

For example, 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 would want 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 enter:

```
1> PATH English:Words ADD
```

Now when you want to access the Spell program, you can simply enter 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 document.

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, enter:

```
1.Testdir> COPY S:Startup-sequence to
:Testdir/Textfile
```

Note The above command should be entered all on one line.

This command makes a copy of the Startup-sequence file in the S: directory, and names it Textfile in the Testdir directory. The original Startup-sequence stays in the S: directory.

12. To verify the COPY command worked, enter:

```
1.Testdir> DIR
```

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 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.

Using TYPE

13. The TYPE command lets you look at the contents of a file. Enter:

```
1.Testdir> 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.

Using RENAME

14. To change the name of a file, use the RENAME command. You must specify both the old name and the new name.

```
1.Testdir> RENAME Textfile Document
```

This command will change the name of the file from Textfile to Document. The contents of the file will not be affected.

Using DELETE

15. To delete a file from the disk, use the DELETE command. Enter:

```
1.Testdir> 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 directory (CD) to a higher-level directory, in this case the original Workbench root directory.

A slash tells the CD command to move up one level in the directory hierarchy.

16. Enter:

```
1> CD /
```

The current directory will become the Workbench root directory.

If you were in a directory such as NewDisk:Testdir/Files, entering CD / would return you to the NewDisk:Testdir directory.

17. To delete the Testdir directory, enter:

```
1> DELETE Testdir
```

The directory will be deleted, but the Testdir.info file will still remain in the Workbench disk's root directory.

18. To delete the Testdir.info file, enter:

```
1> DELETE Testdir.info
```

Your Workbench 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 which allows 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.

Using FORMAT

19. Leave the Workbench disk in the disk drive, and enter:

```
1> FORMAT DEVICE DF0: NAME EmptyDisk
```

Output similar to the following will appear on your screen:

```
Insert disk to be formatted in device DF0:
Press RETURN to begin formatting or CTRL-C to abort:
```

Remove the Workbench disk, and insert a new disk to be formatted into the drive. Press Return to continue.

Caution **Do not press Return until you remove the Workbench disk and insert a blank disk. Otherwise, you could accidentally format your Workbench disk.**

You must enter the DEVICE and NAME keywords with the FORMAT command. The DEVICE keyword specifies the disk drive that the disk is in, and the NAME keyword specifies a new name for the formatted disk.

Using RELABEL

To change the name of a disk, use the RELABEL command. RENAME only works with directories or files.

For example, to change the name of a disk in drive DF0: to NewDisk, you could enter:

```
1> RELABEL DF0: NewDisk
```

Caution **If you have a floppy disk system with only one floppy drive, you should specify the disk by its volume name, not the drive name, DF0:. Using the drive name tells the Amiga to relabel whatever disk is in that drive, and you could accidentally relabel your Workbench disk.**

To change the name of the disk from EmptyDisk to NewDisk using its volume name, begin by removing the newly formatted disk and inserting the Workbench disk.

20. Enter:

```
1> CD EmptyDisk:
```

and follow the System Requests to change disks. Enter:

```
1> RELABEL EmptyDisk: NewDisk
```

and follow the System Requests. The prompt will not automatically change to the new disk name.

Using DISKCOPY

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 describes how to make a copy of your Workbench disk.

21. Remove NewDisk from the drive and insert the Workbench disk. Enter:

```
1>CD Workbenchx.x:
```

where x.x is the version number, i.e., Workbench2.1:.

Now enter:

```
1> DISKCOPY DF0: TO DF0:
```

Output similar to the following should appear on your screen.

```
Place disk to copy from (SOURCE disk) in device DF0
Press RETURN to begin copying or CTRL-C to abort:
```

Continue to follow the system prompts to complete the DISKCOPY. With the DISKCOPY command, you must enter the word TO between the names of the disk drives. If you have two disk drives, you could enter:

```
1> DISKCOPY DF0: TO DF2:
```

When working with two disk drives, you can also use volume names, such as:

```
1> DISKCOPY Workbench: TO NewDisk:
```

Setting the Clock

Use the AmigaDOS DATE command to set the date and time.

Using DATE

22. To display the currently saved date and time, remove the newly copied disk and insert the Workbench disk. Enter:

```
1> DATE
```

The following is a sample of the typical output.

```
Tuesday 16-Jun-92 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 example, to set the date and time to July 22, 1992, 12:34 AM, enter:

```
1> DATE 22-JUL-92 12:34:00
```

The DATE command alone does not save the date and time to the battery back-up clock (if present). If you were to reboot or power off, the date and time would be lost. To save the date and time to the system clock, you must use the SETCLOCK command.

For example, to save the date listed above, you would enter:

```
1> SETCLOCK SAVE
```

Now if the system is re-booted, the 22-JUL-92 12:34:00 setting will remain in effect.

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, respectively.

Using NEWSHELL

24. Enter:

```
1> NEWSHELL
```

A second Shell window should appear on the screen.

Using ENDSHELL

25. To exit that window, make sure it is active, and enter:

```
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 by entering Ctrl+\.

Chapter 5 contains the full capabilities of the commands discussed in this section.

Chapter 3
Using the Shell

Features of the Shell

The Shell allows you to communicate directly with AmigaDOS through a console window. A console window is a text-only interface, which means that it accepts input entered from the keyboard. You cannot use icons in a console window. Special features of the console window are:

- 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. Selecting zoom again will return the window to its original size.
- The System Default Text font, as specified by the Font editor, is used in the Shell window. This must be a non-proportional font, such as Topaz or Courier.
- Workbench background patterns do not appear.

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 arrow keys to fix typing mistakes or repeat commands entered earlier.

Editing

There are several keys and key combinations you can use to edit the command line. They include standard text editing keys, plus several Control key sequences summarized below:

left arrow	Moves cursor one character to the left.
right arrow	Moves cursor one character to the right.
Shift+left arrow	Moves cursor to the beginning of the line.
Shift+right arrow	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-execute 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 arrow keys:

up arrow	Moves backward in the history buffer (earlier lines).
down arrow	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 command 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 entering the command line, or the beginning of it, then pressing Shift+up arrow (or Ctrl+R). For example, if you enter DIR and press Shift+up arrow, you will be returned to the last command entered to perform a DIR of any directory. Pressing Shift+down arrow 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 or restores I/O if * (current window) is used.

Another feature of the Shell is the ability to type ahead. If you begin entering information while output is occurring in a shell window, the output will pause. If you press Return, the output will resume, and the recently entered line will be used as the next line of input. To resume the output without executing the entered line, delete your input. The text will resume scrolling as soon as the last character is erased.

Copying and Pasting

The Shell also allows you to copy and paste information from one console window, such as a Shell or ED window, to the same or another window. For example, 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 re-enter 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.

Now move the pointer to the other console window, and click inside the window at the point where you want to insert the text. Press right Amiga+V, and the highlighted text will be copied to the second window. The text in the first window will remain in memory and can be copied repeatedly.

Customizing the Window

The Shell supports a WINDOW Tool Type in the Shell icon 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.)

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 (unless they were detached). 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 enter information into the window, but there will be no response.

There are three ways to close a Shell window:

- select the close gadget.
- enter 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.

Note In the following section, the angle brackets, <>, indicate that information must be substituted. For example, <name> means that you must enter a name after the alias command. Do not enter the brackets.

An alias takes the form of:

```
ALIAS <name> <string>
```

where <name> is the alias, the name you want to enter at the Shell prompt to execute a command. The <string> is the command line you want to execute. Some example ALIAS commands are:

```
ALIAS d0 DIR DF0:
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 example:

```
ALIAS d0 DIR DF0:
```

lets you enter 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 example, if you are using the alias shown above,

```
ALIAS d0 DIR DF0:
```

entering:

```
d0 System
```

would display the contents of the System directory on the disk in DF0:.

See the "ALIAS" section of Chapter 5 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 followed by a colon (:), 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 below.)

See the "PROMPT" section of Chapter 5 for more detailed information.

Using Escape Sequences

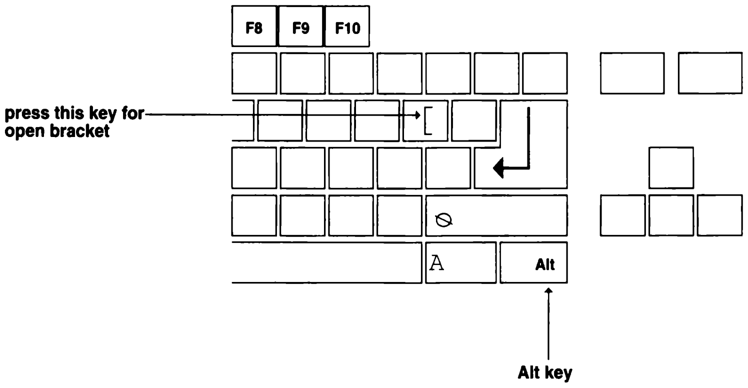
Escape sequences can control how the text appears in a console window, such as the text color, style (bold, italics, underline) and margins. AmigaDOS recognizes standard ANSI X3.64 sequences when they are entered on the command line or embedded in strings. Escape sequences consist of one or more characters, sometimes with a numerical argument, prefaced by the escape character. Spaces are not normally used in the sequence of characters.

The following table 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. Press Esc. Do not enter 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, press the appropriate key.



If your country's keyboard does not have an open bracket key, you must press Alt plus the key shown, regardless of what is shown on the keycap.

Table 3-1. Standard Escape Sequences for Console Windows

Sequence	Action
Escc	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)

Sequence	Action
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.

Running Programs

Most programs can be run from both the Workbench and from the Shell. To run a program from the Shell, you usually enter 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.

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 the MEmacs editor.

```
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 directly on the command line what might otherwise require several menu operations. 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. RUN loads and runs a program in the background. This means that the Shell prompt will return after the program is opened.

For example, if you enter:

```
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 enter:

```
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 example, 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 entering:

```
1> RUN >NIL: MEmacs
```

Using >NIL: redirects (>) the output of the MEmacs editor to a dummy destination (NIL:). It inhibits the display of any output to the Shell window generated by MEmacs, thus allowing you to close the Shell window, without first exiting MEmacs.

Chapter 4
Working with Scripts

A script file, also called a command file, is a text file containing a list of commands. 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 example, 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 example, to rename eight files in one operation you could create the following script:

```
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. Entering 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 halt and request information from the user before continuing, so that variable conditions can be accommodated with a single script. For example, the script below copies six files from a hard drive to a floppy disk. After the six files are copied, the user is asked whether to continue the copy. This allows time to put a new floppy disk into the disk drive if required.

```
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
```

At the "Continue Copy?" prompt, press Y to copy the remaining files to a disk in DF0:. Press N to terminate the copy process.

Several AmigaDOS commands are specifically for use in scripts. These are:

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. Detailed examples of script files can be found in the

EVAL and EXECUTE commands located in Chapter 5. *"AmigaDOS Reference"*.

Condition Flags

Every command sets a condition flag indicating the condition upon which the command will fail. When a command is executed, a return code indicates if a command succeeded or failed. The standard return codes are:

0	The command was successful.
5	Represents a caution indicating that some type of error occurred. The error, however, was not serious enough to abort the command. If the command is part of a script, subsequent commands are 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 example, in the example COPY script described earlier, 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 applications. 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 example, 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. 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. The line is then 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 example, 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 open a second Shell by entering 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 that are recognized by all Shells. Global variables are stored in the ENV: directory. You can use GETENV to display the value associated with a global 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.

A common reason for using an environment variable in a script is to hold string information. If the string is long and tedious to enter, it is easier to substitute a variable for it. Also, if the value for the string changes, it is easier to change the value of the variable than to repeatedly 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 example, you could assign the value 3.14159 to a variable called pi, then use $pi in EVAL expressions. For example:

```
1> EVAL 5 * $pi
15
```

EVAL is an AmigaDOS command that evaluates simple expressions. This command is explained in Chapter 5.

A few applications support the use of environment variables. For example, 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 Extras:Tools/MEmacs
```

Be sure to specify the complete path to MEmacs.

If you are using MORE to look at the contents of the Startup-sequence file, press Shift+E, to be automatically transferred to a 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 (ON) to the screen before being executed, or not (OFF).
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 example, 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 File

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 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.

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.

Any AmigaDOS command can appear in a startup script. Be sure to read Chapter 5 for complete specifications about the individual commands before making changes.

Editing the Startup-sequence File

Caution **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 to 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 5. Test 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 are not referencing something that has not yet

been created, assigned, or given a valid path. For example, if you insert a command to copy a file to the T: directory, and the T: directory has not yet been created or assigned, you will receive an error message.

- 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, but appears in the script as a comment to remind you of what you are trying to accomplish. For example:

  ```
  ASSIGN T: RAM:T ; set up 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. Enter WHY at the prompt after the error appears to get a better explanation of the error.

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 from 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:
NewShell CON:0/0/640/200/AmigaShell/CLOSE
```

You can substitute cd RAM: for cd SYS:. You can also 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 an external disk drive, you can add additional paths and logical device names to your User-startup file. For example, if you always boot with the Extras disk in DF2:, you can add the following line:

```
PATH Extras: ADD
```

If you always boot with a C language Source disk (with Include, and Lib directories) in DF2:, it may be convenient to assign logical names to these directories. If your scripts always refer to these logical names, you will only have to change the following ASSIGN statements when you change your system or configuration. For example, if the disk is named MySrc, you can 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. The following 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, these commands will be executed.

For more information on working around the limitations of a single floppy-based system, see Appendix C. *"Single Floppy Disk Systems."*

Chapter 5
AmigaDOS Reference

This chapter gives complete specifications of all the AmigaDOS commands, including:

- command conventions, an explanation of the symbols and abbreviations used in the command descriptions.
- specifications for each command, including Workbench and Preferences programs.

Command Conventions

When you invoke an AmigaDOS command, you usually do more than enter 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 enter:

```
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 entered:

```
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:

Format	All the arguments and options accepted by a command.
Template	A built-in reminder of the command's format. The template is embedded in the program's code. If you enter a command followed by a question mark (DIR ?), the template will appear on the screen.
Purpose	A short explanation of the command's function.
Path	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 ROM, and the Workbench programs.
Specification	A description of the command and all of its arguments.
Examples	When examples are given, the command and any screen output are displayed in a courier typeface to distinguish them from the main text. A generic 1> prompt indicates what should be entered 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 entered as part of the command.

< >	Angle brackets enclose arguments that must be provided. For example, <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 from 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][DATES][NOPRO][COM]
[NOREQ]
```

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 specify only one destination.

The [ALL], [QUIET], [CLONE], [DATES], [NOPRO], [COM], and [NOREQ] arguments are optional.

The [BUF | BUFFER = <n>] argument is optional. If given, you can use either BUF or BUFFER and a numerical argument. For example, 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 enter 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 entered 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 enter 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. 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 enter 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 entered as part of the command.

The Template for the COPY command is shown below:

```
FROM/M,TO/A,ALL/S,QUIET/S,BUF=BUFFER/K/N,
CLONE/S,DATES/S,NOPRO/S,COM/S,NOREQ/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 must be given (/K) if the BUF or BUFFER argument is used. Both BUF and BUFFER are acceptable keywords (=).

Generally you can link keywords and their arguments with an equal sign (=), which serves to ensure correct assignments in complex cases. For example, BUF=20.

Public Screens - The PUBSCREEN Option

The Amiga environment supports a great number of screens as described in *Using the Amiga Workbench*. Applications that create screens can mark screens as public, which enables other applications and utilities to open their windows on the same screen.

Most of the programs described in this chapter that have a graphical user interface allow you to open their windows on a public screen. Whenever a command includes PUBSCREEN/K in its template, it means the program can open its windows on a public screen. For example, the template for the Input preferences editor is:

```
FROM,EDIT/S,USE/S,SAVE/S,PUBSCREEN/K
```

To open the Input preferences editor on a public screen, enter:

```
1> INPUT PUBSCREEN "screen name"
```

where "screen name" is the public name of the screen to open on.

The public name of a screen is an internal name given by an application. The name has no direct relation to the screen title printed in the screen's title bar. Often a screen's name will be the same as the application's name. Consult the documentation that comes with your application to find out if it opens public screens and what their names are.

Command Specifications

ADDBUFFERS

Format:	ADDBUFFERS <drive> [<n>]
Template:	DRIVE/A,BUFFERS/N
Purpose:	To instruct 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 512KB 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. (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 disk storage in your system.

If only the <drive> argument is specified, ADDBUFFERS displays the number of buffers currently allocated for that drive.

Example 1:

```
1> ADDBUFFERS DF0:
DF0: has 5 buffers
```

Displays the number of buffers currently allocated to drive DF0:.

Example 2:

```
1> ADDBUFFERS DF1: 25
DF1: has 30 buffers
```

Adds 25 buffers to drive DF1: and displays the total.

ADDDATATYPES

Format:	ADDDATATYPES {FILES} [QUIET] [REFRESH]
Template:	FILES/M,QUIET/S,REFRESH/S
Purpose:	To build a list of data types that datatypes.library can understand.
Path:	C:ADDDATATYPES

Specification:

ADDDATATYPES is used by datatypes.library to build a list of data types that it can understand. DataType descriptors are stored in DEVS:DataTypes.

ADDDATATYPES can also be called by application Installation scripts to add their own data types to the list. Basically, ADDDATATYPES is not a user program.

The FILES option specifies the name(s) of the DataType descriptors to add to the list of DataType descriptors.

The QUIET option, if specified, will suppress error and output messages.

The REFRESH option, if specified, will scan the DEVS:DataTypes directory for new or changed DataType descriptors.

ADDDATATYPES is valid on Workbench 3.0 level software.

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 entered at the beginning of the command line. You can enter arguments after the alias. However, you cannot create an alias to represent a series of command arguments. For example, in the following comand line:

```
1>LIST Picdir TO RAM:filelist
LFORMAT="SYS:Utilities/display %f%n"
```

you could not replace the LFORMAT argument with an alias. You can substitute a filename or other instruction within an alias by placing square brackets ([ ]) in the <string>. Any argument entered 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.

Example 1:

```
1> ALIAS d1 DIR DF1:
```

Entering d1 results in a directory of the contents of the disk in DF1:, just as if you had entered DIR DF1:.

Example 2:

```
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 entered:

```
1> hex Myfile
```

the contents of Myfile would be displayed in hexadecimal format with line numbers.

See also: UNALIAS

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 string specified by <prompt> to the current window then wait for keyboard input. Valid keyboard responses are Y (yes), N (no), and Return (no). If Y is pressed, ASK sets the condition flag to 5 (WARN). If N or Return 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. 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 EXISTS 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.

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.

It is not necessary for the assigned name to retain the name of the directory nor is it necessary for it to be uppercase. For example, both CLIPS and Clips can be assigned to the Ram Disk:Clipboards directory.

The PATH option creates a "non-binding" ASSIGN. A non-binding ASSIGN acts like a DEFERred ASSIGN except that it is re-evaluated 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 example, 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 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.

Example 1:

```
1> ASSIGN FONTS: MyFonts:Fontdir
```

assigns the FONTS: directory to Fontdir on MyFonts:.

Example 2:

```
1> ASSIGN LIST
Volumes:
Ram Disk [Mounted]
Workbench [Mounted]
MyFonts [Mounted]

Directories:
LOCALE      Workbench:Locale
KEYMAPS     Workbench:Devs/Keymaps
PRINTERS    Workbench:Devs/Printers
REXX        Workbench:S
CLIPS       Ram Disk:Clipboards
ENV         Ram Disk:Env
T           Ram Disk:T
ENVARC      Workbench:Prefs/Env-Archive
SYS         Workbench:
C           Workbench:C
S           Workbench:S
L           Workbench:L
FONTS       MyFonts:Fontdir
DEVS        Workbench:Devs
LIBS        Workbench:Libs

Devices:
PIPE AUX RAM CON
RAW PAR SER PRT DF0 DF1
```

shows a list of all current assignments.

Example 3:

```
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.

Example 4:

```
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.

Example 5:

```
1> ASSIGN DEVS: REMOVE
```

removes the DEVS: assignment from the system.

Example 6:

```
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:>.

Example 7:

```
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 it is a non-binding ASSIGN.

Example 8:

```
1> ASSIGN LIBS: ZCad:Libs ADD
```

adds ZCad:Libs to the list of directories assigned as LIBS:.

Example 9:

```
1> ASSIGN LIBS: ZCad:Libs REMOVE
```

removes ZCad:Libs from the list of directories assigned as LIBS:.

AUTOPOINT

Format:	AUTOPOINT [CX_PRIORITY <priority>]
Template:	CX_PRIORITY/N/K
Purpose:	To automatically select any window the pointer is over.
Path:	Extras: 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 <priority> argument sets the priority of AutoPoint in relation to all the other Commodity Exchange programs. (This is the same as entering a CX_PRIORITY=<priority> Tool Type in the icon's Information window.) All the Commodity Exchange programs are set to a default priority of 0. If you specify a <priority> 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, press Ctrl+C or use the BREAK command.

Example:

```
1> AUTOPOINT
```

starts the AutoPoint program.

AVAIL

Format:	AVAIL [CHIP I FAST I 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 of memory, 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 FLUSH option causes all unused libraries, devices, and fonts to be expunged from memory.

Example 1:

```
1> AVAIL
Type     Available    In-Use    Maximum    Largest
chip        233592    282272     515864      76792
fast        341384    182896     524280     197360
total       574976    465168    1040144     197360
```

A complete summary of system RAM is displayed.

Example 2:

```
1> AVAIL CHIP
233592
```

The number of free bytes of CHIP RAM is displayed.

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 automatically configured by the expansion library. The 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.

BLANKER

Format:	BLANKER [CX_PRIORITY <priority>] [CX_POPKEY <key>] [CX_POPUP=<yes | no>] [SECONDS <timeout>] [CYCLECOLORS <yes | no>] [ANIMATION <yes | no>]
Template:	CX_PRIORITY/N/K,CX_POPKEY/K,CX_POPUP/K, SECONDS/N/K,CYCLECOLORS/K,ANIMATION/K
Purpose:	To cause the monitor screen to go blank if no input has been received within a specified period of time.
Path:	Extras: 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 argument allows you to specify the number of seconds that must pass. The acceptable range is from 1 to 9999. Default is 60 seconds.

CX_POPKEY 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 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.

If CYCLECOLORS=yes is specified, Blanker will cycle through a series of colors. If CYCLECOLORS=no is specified, Blanker will display the default color.

If ANIMATION=yes is specified, the screen will display a series of random splines. If ANIMATION=no is specified, a blank screen will appear.

To exit Blanker when it has been started through the Shell, press Ctrl+C or use the BREAK command.

Example 1:

```
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.

Example 2:

```
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. Use the STATUS command to display the currrent process numbers. 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, only the Ctrl+C flag is set.

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. Use the STATUS command to display the current process numbers.

Ctrl+F is used by programs that open windows. Sending these programs a Ctrl+F signal will cause them to activate their window and bring it to the front of all windows. Not all programs respond to Ctrl+F.

Example 1:

```
1> BREAK 7
```

sets the Ctrl+C attention flag of process 7. This is identical to selecting process 7 and pressing Ctrl+C.

Example 2:

```
1> BREAK 5 D
```

sets the Ctrl+D attention flag of process 5.

See also: STATUS

CALCULATOR

Format:	CALCULATOR [PUBSCREEN <public screen name>] [TAPE <window>]
Template:	PUBSCREEN,TAPE/K
Purpose:	To provide an on-screen calculator.
Path:	Extras:Tools/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.

TAPE allows you to create a Calculator window of a specific size in which to display your input and output. The specification is in the form of:

```
TAPE=RAW: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 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.)

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, enter 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, i.e., CD :.

CD also supports pattern matching. If more than one directory matches the given pattern, an error message is displayed.

Example 1:

```
1> CD DF1:Work
```

sets the current directory to the Work directory on the disk in drive DF1:.

Example 2:

```
1> CD SYS:Com/Basic
```

makes the subdirectory Basic in the Com directory the current directory.

Example 3:

```
1> CD //
```

moves up two levels in the directory structure.

Example 4:

```
1> CD SYS:Li#?
```

uses the #? pattern to match with the Libs: directory.

CHANGETASKPRI

Format:	CHANGETASKPRI <priority>PROCESS [<process number>]
Template:	PRI=PRIORITY/A/N,PROCESS/K/N
Purpose:	To change the priority of a currently running process.
Path:	C:CHANGETASKPRI

Specification:

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 number> inherit its priority.

Use the STATUS command to display the current process numbers.

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

CLICKTOFRONT

Format:	CLICKTOFRONT [CX_PRIORITY <priority>] [QUALIFIER <qualifier>]
Template:	CX_PRIORITY/N/K,QUALIFIER/K
Purpose:	To bring a window to the front of the screen.
Path:	Extras: Tools/Commodities/ClickToFront

Specification:

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.

The CX_PRIORITY <priority> argument sets the priority of ClickToFront in relation to all the other Commodity Exchange programs. (This is the same as entering a CX_PRIORITY=<priority> Tool Type in the icon's Information window.) All the Commodity Exchange programs are set to a default priority of 0. If you specify a <priority> value higher than 0, ClickToFront will take priority over any other Commodity Exchange program.

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.)

QUALIFIER allows you to specify a qualifier key that must be pressed while double-clicking in the window you want to bring to the front of the screen. The acceptable key arguments are:

Lalt	Left Alt
Left_Alt	Left Alt
Ralt	Right Alt
Right_Alt	Right Alt
Ctrl	Ctrl
Control	Ctrl
None	No key

To exit ClickToFront when it has been started from the Shell, press Ctrl+C, or use the BREAK command.

CLOCK

Format:	CLOCK [DIGITAL] [<LEFT>] [<TOP>] [<WIDTH>] [<HEIGHT>] [24HOUR] [SECONDS] [DATE] [<FORMAT>] [PUBSCREEN <public screen name>]
Template:	DIGITAL/S,LEFT/N,TOP/N,WIDTH/N,HEIGHT/N, 24HOUR/S,SECONDS/S,DATE/S,FORMAT/N, PUBSCREEN/K
Purpose:	To provide an on-screen clock.
Path:	SYS:Utilities/Clock

Specification:

CLOCK allows you to display an on-screen clock. 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 example, 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 entered two numbers, the clock would interpret them as the LEFT and TOP positions.

WIDTH and HEIGHT are not available if you use the DIGITAL option. You cannot change the size of the digital clock, althougb you can specify its position.

If the SECONDS option is specified, the seconds are displayed.

If the DATE option is specified, the date is displayed.

Example 1:

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, enter:

```
1> CLOCK 75 75 300 100
```

Example 2:

To use the SECONDS, DATE and 24HOUR options, enter:

```
1> CLOCK SECONDS DATE 24HOUR
```

Example 3:

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), enter:

```
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:	Extras: 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 entered.
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:	Extras: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>		*<screentype>*	
Specifies the depth of the test screen:		**Specifies the resolution of the test screen:**	
1	2 colors	**0**	320 x 200 pixels
2	4 colors	**1**	320 x 400 pixels
3	8 colors	**2**	640 x 200 pixels
4	16 colors	**3**	640 x 400 pixels
5	32 colors		

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 (High Res) resolution.

CONCLIP

Format:	CONCLIP [UNIT <unit number>] [OFF]
Template:	UNIT/N,OFF/S
Purpose:	To move data between the console.device, the clipboard.device and CON:
Path:	C:CONCLIP

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. Once 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 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 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 no 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/M,TO/A,ALL/S,QUIET/S, BUF=BUFFER/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 100KB 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, rather than the date and time the file was created or last revised. 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 or last revised date (whichever is most recent) 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 creation (or last revised) 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 example, 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.

Example 1:

```
1> COPY File1 TO :Work/File2
```

copies File1 in the current directory to the File2 directory in the Work directory.

Example 2:

```
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.

Example 3:

```
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.

Example 4:

```
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.

Example 5:

```
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

Format: CPU [CACHE] {NOCACHE} [BURST] [NOBURST] [DATACACHE] [DATABURST] [NODATACACHE] [NODATABURST] [INSTCACHE] [INSTBURST] [NOINSTCACHE] [NOINSTBURST] [FASTROM] [NOFASTROM] [NOMMUTEST] [CHECK 68010 | 68020 | 68030 | 68040 | 68881 | 68882 | 68851 | M MU | FPU] [TRAP] [NOTRAP] [COPYBACK] [NOCOPYBACK] [EXTERNALCACHE] [NOEXTERNALCACHE]

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,COPYBACK/S, NOCOPYBACK/S,EXTERNALCACHE/S, NOEXTERNALCACHE/S,FASTROM/S, NOFASTROM/S,TRAP/S,NOTRAP/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 on a 68000 Amiga.

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.

Many options only work with certain members of the 680X0 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. The CPU options, outlined below, specify the types of memory to be used. If mutually exclusive options are specified, the safest option is used.

Availability of the options listed below depends on availability of required hardware.

CACHE	Turns on all caches.
NOCACHE	Turns off all caches.
BURST	Turns on burst mode for both data and instructions.
NOBURST	Turns off burst mode for data and instructions.
DATACACHE	Turns on data cache.
NODATACACHE	Turns off data cache.
DATABURST	Turns on burst mode for data.
NODATABURST	Turns off burst mode for data.
INSTCACHE	Turns on instruction cache.
INSTBURST	Turns on burst mode for instruction.
NOINSTCACHE	Turns off instruction cache.
NOINSTBURST	Turns off burst mode for instructions.
FASTROM	If supported by the MMU, 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 settings to be changed without checking to see if MMU is currently in use.
TRAP	Developer-specific.
NOTRAP	Developer-specific.
COPYBACK	Turns on cache copyback.
NOCOPYBACK	Turns off cache copyback.
EXTERNALCACHE	Turns on external cache.
NOEXTERNALCACHE	Turns off external cache.

The CHECK option, when given with a keyword (68010, 68020, 68030, 68040, 68881, 68882, or 68851, MMU, FPU) checks for the presence of the keyword.

Examples:

```
1> CPU
System: 68030 68881 (INST: Cache Burst) (DATA: Cache
NoBurst)

1> CPU NoDataCache FastROM
System: 68030 68881 FastROM (INST: Cache Burst)
(DATA: NoCache NoBurst)

1> CPU NoBurst DataCache NoInstCache
System: 68030 68881 (INST: NoCache NoBurst) (DATA:
Cache NoBurst)
```

CROSSDOS

Format:	CROSSDOS [CX_PRIORITY <priority>] [CX_POPKEY <key>] [CX_POPUP <yes | no>]
Template:	CX_PRIORITY/N/K,CX_POPKEY/K,CX_POPUP/K,
Purpose:	To set text filter and conversion options.
Path:	Extras: Tools/Commodities/CrossDOS

Specification:

CROSSDOS lets you read from and write to MS-DOS formatted disks using your standard Amiga drives. This makes it simple to transfer information such as text, font, database, and graphics files between Amiga and MS-DOS computers. Use the CROSSDOS command to set text filter and conversion options during this transfer.

The standard DOS drivers for CrossDOS are PC0: and PC1:, which correspond to DF0: and DF1:. These two drivers allow an Amiga floppy drive to read from and write to 720KB MS-DOS disks. CrossDOS drivers still handle your Amiga disks normally.

CX_POPKEY 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 CrossDOS window from opening. By default the program window opens when the command is invoked.

CX_PRIORITY sets the priority of CrossDOS in relation to all other Commodity Exchange programs. All the Commodity Exchange programs are set to a default priority of 0.

For complete information on CrossDOS, see *Using the Amiga Workbench* manual.

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 number or the first 3 letters of the month (in English) must be used, as well as the last two digits of the year.

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 cannot specify a day name to change the date to more than seven days into the future. 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.

Adjustments made with DATE only change the software clock. They will not survive past power-down. To set the battery back-up hardware clock from the Shell, you must set the date then use SETCLOCK SAVE.

Example 1:

```
1> DATE
```

displays the current date and time.

Example 2:

```
1> DATE 6-sep-82
```

sets the date to September 6, 1982. The time is not reset.

Example 3:

```
1> DATE tomorrow
```

resets the date to one day ahead.

Example 4:

```
1> DATE TO Fred
```

sends the current date to the file Fred.

Example 5:

```
1> DATE 23:00
```

sets the current time to 11:00 p.m.

Example 6:

```
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] [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 files. 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.

Caution 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.

Example 1:

```
1> DELETE Old-file
```

deletes the file named Old-file in the current directory.

Example 2:

```
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).

Example 3:

```
1> DELETE T#?/#?(1|2)
```

deletes all the files that end in 1 or 2 in directories that start with T.

Example 4:

```
1> DELETE DF1:#? ALL FORCE
```

deletes all the files on DF1:, even those set as not deletable.

See also: PROTECT

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.

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. DEL does not refer to the Del key; enter the letters D, E, and L.
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.

The COMMAND option allows almost any AmigaDOS command to be executed during the interactive directory list. When you want to issue a command, enter C (or COMMAND) at the question mark prompt. DIR will ask you for the command. Enter 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 example,

```
1>? C "type prefs.info hex"
```

is equivalent to pressing Q to exit interactive listing mode and return to a regular Shell prompt, and entering:

```
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.

Example 1:

```
1> DIR Workbench:
```

displays a list of the directories and files on the Workbench disk.

Example 2:

```
1> DIR MyDisk:#?.memo
```

displays all the directories and files on MyDisk that end in .memo.

Example 3:

```
1> DIR Extras: ALL
```

displays the complete contents of the Extras disk — all directories, all subdirectories and all files, including those in the subdirectories.

Example 4:

```
1> DIR Workbench: DIRS
```

displays only the directories on Workbench.

Example 5:

```
1> DIR Workbench: INTER
```

provides an interactive list of the contents of the Workbench disk.

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.

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 enter:

```
1> DISKCHANGE DF2:
```

AmigaDOS will then recognize the new disk, and you can proceed.

DISKCOPY

Format:	DISKCOPY [FROM] <device> TO <device> [NAME <name>] [NOVERIFY] [MULTI]
Template:	FROM/A,TO/A,NAME/K,NOVERIFY/S,MULTI/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 <device> parameters specify the name of the disk devices to copy from and to copy to, for example, DF0: or DF1:.

A floppy disk containing an altered icon can be displayed from the Workbench by removing the disk from the drive and reinserting it.

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.

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.

Example 1:

```
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:

Example 2:

```
1> DISKCOPY DF0: to DF2: NAME NewDisk NOVERIFY
```

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.

DISPLAY

Format:	DISPLAY {<filename> | FROM <filelist>} [OPT mlbpaenv 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 enter 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 that the picture is in Extra Halfbrite mode. 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. A CAMG chunk is part of an IFF file that describes in which viewmode the picture should be displayed.
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.

The t=<n> argument specifies the number of seconds the IFF file will be displayed. This allows for automatic advancing through files.

Example 1:

```
1> DISPLAY file1 file2 file3
```

displays the files in the order given. To advance from one file to the next, press Ctrl+C.

Example 2:

```
1> DISPLAY FROM Scriptlist
```

displays the files listed in the Scriptlist file. Pressing Ctrl+C will advance to the next file.

Example 3:

```
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.

Example 4:

```
1> DISPLAY FROM Scriptlist OPT t=5
```

displays each file in the Scriptlist file for five seconds.

DISPLAY is replaced by MultiView in Workbench 3.0 level software.

ECHO

Format:	ECHO [<string>] [NOLINE] [FIRST <n>] [LEN <n>]
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. By default this is the screen, but it could be to any 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 from which 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 example, 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>
```

ED

Format:	ED [FROM] <filename> [SIZE <n>] [WITH <filename>] [WINDOW] [TABS <n>] [WIDTH <n>] [HEIGHT <n>]
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 6, "*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 6, "*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, called Display, contained the following block:

```
IF exists <name>
   TYPE <name> NUMBER
ELSE
   ECHO "<name> is not in this directory"
ENDIF
```

To execute this script, you could enter:

```
1> EXECUTE Display work/prg2
```

If the work/prg2 file can be found in the current directory, the TYPE <name> NUMBER 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:

ENDSHELL ends a Shell process.

The Shell process can also be ended by ENDCLI or clicking on the Close gadget.

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 example, 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, are truncated (cut off). Decimals are not allowed, evaluation stops at the decimal point.

<Value1> and <value2> may be 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 (').

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 below:

Table 5-1. EVAL Operations

Operation	Symbol
addition	+
subtraction	-
multiplication	*
division	/
modulo	mod
AND	&
OR	|
NOT	~
left shift	lsh
right shift	rsh
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.

Example 1:

```
1> EVAL 64 / 8 + 2
10
```

Example 2:

```
1> EVAL 0x5f / 010 LFORMAT="The answer is %X4*N"
The answer is 000B
```

This divides hexadecimal 5f (95) by octal 10 (8), yielding 000B, the integer portion of the decimal answer 11.875. (The 1> prompt would have appeared immediately after the 000B if *N had not been specified in the LFORMAT string.)

Example 3:

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
DELETE ENV:Loop T:Qwe{$$} ; clean up
ECHO "done"
```

If you were to enter:

```
1> EXECUTE Loop 5
```

the following results would be displayed:

```
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>, which makes the question mark at the end of the line necessary. <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 <yes | no>] [CX_PRIORITY <priority>]
Template:	CX_PRIORITY/N/K,CX_POPKEY/K,CX_POPUP/K
Purpose:	To monitor and control the Commodity Exchange programs.
Path:	Extras:Tools/Commodities/Exchange

Specification:

EXCHANGE is a Commodity Exchange program that monitors and controls all the other Commodity Exchange programs. CX_POPKEY 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 will keep the Exchange window from opening.

CX_PRIORITY 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 exit EXCHANGE once it has been started from the Shell, press Ctrl+C or use the BREAK command.

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:	FILE/A
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 entered 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 enter 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.

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 example, if the first line of a script is:

```
.KEY filename/A, TO/K
```

you must specify a filename variable. The TO variable is optional, but if specified the TO keyword must be used. For example:

```
1> EXECUTE Script Textfile TO 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 .KET <ch> 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 example:

```
.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 entering a dot (.), followed by a space, then the comment.

.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.

Example 1:

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 entered the following commands at the keyboard:

```
1> RUN COPY Test/Prg TO PRT: +
1> ECHO "Printing of Test/Prg done"
```

Example 2:

Another example, Display, uses more of the features described above:

```
.Key name/A
IF EXISTS <name>
TYPE <name> NUMBER  ;if 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 example, 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}
Template:	/N/M
Purpose:	To print the messages(s) for the specified error code(s).
Path:	Internal

Specification:

FAULT prints the messages corresponding to the error numbers supplied. Any number of error numbers can be specified at once. If several error numbers are given with FAULT, they must be separated by spaces.

Example:

If you received the error message:

```
Error when opening DF1:TestFile 205
```

and needed more information, you would enter:

```
1> FAULT 205
FAULT 205: object not found
```

This tells you that the error occurred because TestFile could not be found on DF1:.

A complete list of error messages appears in Appendix A.

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 original 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.

Example 1:

```
1> FILENOTE Sonata "allegro non troppo"
```

attaches the filenote "allegro non troppo" to the Sonata file.

Example 2:

```
1> FILENOTE Toccata "*"presto*""
```

attaches the filenote ""presto"" to the Toccata file.

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.

FIXFONTS should be used whenever you make changes in the FONTS: directory, for example, copying new font files or deleting single font sizes.

Example:

```
1> FIXFONTS
```

FKEY

Format:	FKEY [CX_PRIORITY <priority>] [CX_POPKEY <key>] [CX_POPUP <yes | no>]
Template:	CX_PRIORITY/N/K,CX_POPKEY/K,CX_POPUP/K,
Purpose:	To assign text to function and shifted function keys.
Path:	Extras:Tools/Commodities/FKey

Specification:

FKEY allows you to assign functions to keys, eliminating the need for repetitive typing.

FKEY lets you assign any of eight commands to any key sequence that you can enter. The Defined Keys scrolling list shows all the currently defined key sequences. The New Key and Delete Key gadgets let you add and remove key sequences.

The Command cycle gadget lets you pick a command for the current key sequence. The possible commands are the following:

Cycle Windows	Brings the rearmost application window on the Workbench screen to the front of the display and activates it. This only affects application windows opened by tools or projects, such as Clock. Disk and drawer windows are not affected.
Cycle Screens	Brings the rearmost screen to the front of the display.
Enlarge Window	Enlarges the active window to its maximum size, taking into account the edges of the screen.
Shrink Window	Shrinks the active window to its minimum size.
Toggle Window Size	Zooms the active window just as if you had selected the window's zoom gadget. It also works on windows that only have a zoom gadget and no sizing gadgets.
Insert Text	When the key sequence is entered, the specified string is inserted instead. The string to insert is specified in the Command Parameters gadget.

Run Program	Lets you run a program by pressing any key sequence. The program name and its arguments are specified in the Command Parameters gadget.
Run ARexx Script	Lets you run an ARexx script by pressing any key sequence. A script name and its arguments are specified in the Command Parameters gadget. Putting quotes around the script name turns it into an ARexx script.

FONT

Format:	FONT [FROM <filename>] [EDIT] [USE] [SAVE] [PUBSCREEN <public screen name>]
Template:	FROM,EDIT/S,USE/S,SAVE/S,PUBSCREEN/K
Purpose:	To specify the fonts used by the system.
Path:	Extras:Prefs/Font

Specifications:

FONT with no arguments or with the EDIT argument opens the Font 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 Font editor. For example, if you have saved a special configuration of the Font 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.

FORMAT

Format:	FORMAT DEVICE | DRIVE <device> NAME <name> [<OFS/FFS>] [<INTERNATIONAL | NOINTERNATIONAL>] [NOICONS] [QUICK]
Template:	DEVICE=DRIVE/K/A,NAME/K/A,OFS/S | FFS/S, INTL=INTERNATIONAL/S, NOINTL=NOINTERNATIONAL/S,NOICON/S, QUICK/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 DEVICE 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.

The OFS option forces the disk to be formatted using the Old File System. This is the default for floppy disks. The FFS option causes the disk to be formatted using the Fast File System. This can provide for faster operation than OFS disks, but the resulting disks cannot be shared with Amigas with system software releases prior to 2.0. The INTERNATIONAL option forces the disk to be formatted using the international versions of the file systems. These international versions deal correctly with upper and lower letter case conversions of international characters in filenames. The NOINTERNATIONAL option forces the non-international equivalent to OFS NOINTERNATIONAL for a floppy device, and FFS NOINTERNATIONAL for a credit card device.

Example 1:

```
1> FORMAT DRIVE DF0: NAME EmptyDisk
```

formats the disk in drive DF0:, erases any data, and names the disk EmptyDisk.

Example 2:

```
1> FORMAT DRIVE DF2: NAME NewDisk QUICK
```

Reformats, or erases, a disk that already contains data.

FOUNTAIN

Format:	FOUNTAIN [VALIDATE]
Template:	VALIDATE/S
Purpose:	To manage use of Intellifont outline fonts.
Path:	SYS: System/Fountain

Specification:

FOUNTAIN, which is in the System drawer on the Workbench disk, manages Intellifont outline fonts on your Amiga. With FOUNTAIN, you can install new outline fonts on your system, specify new sizes for existing fonts, and delete fonts that are no longer needed. You can create bitmap versions of any size outline font that can be used in applications that do not support outline fonts directly.

FOUNTAIN is valid on Workbench 2.1 level software.

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
Extras: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
Extras: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:	Extras:Tools/GraphicDump

Specification:

GRAPHICDUMP sends a dump of the frontmost screen to the printer about ten seconds after issuing the command. The ten second delay allows the user to bring the wanted screen to the front. 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 proportions of the screen are maintained.

To specify exact dimensions, substitute the absolute width in dots for <xdots> and the absolute height for <ydots>.

Example 1:

```
1> GRAPHICDUMP SMALL
```

will produce a printout of the frontmost screen that is about one-half the total width allowed by the printer.

Example 2:

```
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:	Extras: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] [PUBSCREEN <public screen name>]
Template:	FROM,EDIT/S,USE/S,SAVE/S,PUBSCREEN/K
Purpose:	To specify parameters used by the Workbench.
Path:	Extras: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 example, 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/IControl.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 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.

Several Tool Types can be specified in the icons for the script files. The WINDOW Tool Type lets you provide an alternate window specification for the input/output window. By default, the window's specification is :

```
WINDOW=CON:0/50//80/IconX/AUTO/WAIT/CLOSE
```

This has the effect of opening a window only if there actually is some output performed by the script. If a window does open, it will remain opened after the script terminates, until the user clicks on its close gadget.

The WAIT Tool Type lets you specify the number of seconds the input/output window should remain open after the script terminates. Using this option will cause the default input/output window to not have a close gadget. There is also a DELAY Tool Type which works in a very similar way, except that its parameter is in 1/50th of a second, instead of in seconds.

The STACK Tool Type lets you specify the number of bytes to use for stack for the script execution. If this Tool Type is not provided, the default 4096 bytes is used.

Finally, the USERSHELL Tool Type lets you cause the script file to be run using the current User Shell instead of the normal ROM Shell. You must specify USERSHELL=YES for this to happen. User Shells are third party shells that you can purchase and install in your system to replace the standard shell environment that comes with the operating system.

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.

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 <b>	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.

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.

Example 1:

```
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.

Example 2:

```
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

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 floppy disk drive and hard disk 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   Workbench
DF1:    879K  418 1140  24%   0  Read/Write  Text-6

Volumes available:
Workbench [Mounted]
Text-6 [Mounted]
```

INITPRINTER

Format: INITPRINTER

Template: (none)

Purpose: To initialize a printer for print options specified in the Preferences editors.

Path: Extras: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. The printer gets initialized automatically on first access, but if you switched it off in the interim, you have to initialize the printer again with INITPRINTER.

Example:

```
1> INITPRINTER
```

INPUT

Format: INPUT [FROM <filename>] [EDIT] [USE] [SAVE] [PUBSCREEN <public screen name>]

Template: FROM,EDIT/S,USE/S,SAVE/S,PUBSCREEN/K

Purpose: To specify different speeds for the mouse and keyboard and to select a national keyboard.

Path: Extras: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 example, 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: | CC0:> [NOBOOT] [CHECK] [FFS]
Template:	DRIVE/A,NOBOOT/S,CHECK/S,FFS/S
Purpose:	To write the boot block to a formatted floppy disk, or credit card, specifying whether it should be bootable.
Path:	C:INSTALL

Specification:

INSTALL clears a floppy disk's or credit card's boot block area and writes a valid boot block onto the media.

The NOBOOT option removes the boot block from an AmigaDOS disk or card, making it not bootable.

The CHECK option checks for valid boot code. It reports whether a disk or card is bootable or not and whether standard Commodore-Amiga boot code is present on the media. The condition flag is set to 0 if the boot code is standard (or the disk or card isn't bootable), 5 (WARN) otherwise.

The FFS switch is ignored. It remains part of the template to ensure compatibility with earlier scripts and programs.

Example 1:

```
1> INSTALL DF0: CHECK
No bootblock installed
```

indicates that there is a non-bootable floppy in DF0:.

Example 2:

```
1> INSTALL DF0:
```

makes the disk in drive DF0: a bootable disk.

Example 3:

```
1> INSTALL DF0: CHECK
Appears to be FFS bootblock
```

indicates that there is an FFS floppy in DF0:.

INTELLIFONT

Format:	INTELLIFONT [VALIDATE]
Template:	VALIDATE/S
Purpose:	To manage use of Intellifont outline fonts.
Path:	SYS: System/Intellifont

Specification:

INTELLIFONT, which is in the System drawer on the Workbench disk, manages Intellifont outline fonts on your Amiga. With INTELLIFONT, you can install new outline fonts on your system, specify new sizes for existing fonts, and delete fonts that are no longer needed. You can create bitmap versions of any size outline font that can be used in applications that do not support outline fonts directly.

INTELLIFONT is valid on Workbench 3.0 level software and replaces the predecessor program FOUNTAIN.

IPREFS

Format: IPREFS [QUIT]

Template: QUIT/S

Purpose: To communicate Preferences information stored in the individual editor files to the Workbench and Intuition.

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.

QUIT will halt all IPREFS operations. Since IPREFS runs in the background, there is no reason to use this option.

JOIN

Format: JOIN [FILE] {<file | pattern>} AS | TO <filename>

Template: FILE/M/A,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:	Extras: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.

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. 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 | filename>}] [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,----rwed for readable/writable/executable/delete-able. See the PROTECT command for more on protection bits.
date and time	The date and time the file was created or last changed.
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:

KEYS	Displays the block number of each file header or directory. This is mostly useful to programmers.
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 the specified date.
UPTO <date>	Lists only files created on or before the specified 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 script file. 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:

	Substituted with each occurrence			
Occurrences of %S	**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 options allow you to specify fields to be printed in the LFORMAT output. These options 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.
%E	Prints just the file extension.
%K	Prints the file key block.
%L	Prints the length of file in bytes.
%M	Prints filename only, omitting any extension.
%N	Prints the name of the file.
%P	Prints the file parent path.
%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. To specify left justification, place a minus sign (-) before the length specifier. Otherwise, the information displayed will be right justified.

Example 1:

```
> LIST Dirs
Monitors          Dir   ----rwed  27-Jun-90    11:43:59
T                 Dir   ----rwed  16-Jul-90    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 typical output is shown above.)

Example 2:

```
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 Libs.file in RAM:.

Example 3:

```
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.

Example 4:

```
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. For example, if the current directory is S:, the contents of Scriptnotes as produced by this command might look like this:

```
filenote s:HDBackup.config Testnote
filenote s:DPat Testnote
filenote s:Ed-startup Testnote
filenote s:PCD Testnote
filenote s:Shell-startup Testnote
filenote s:SPat Testnote
filenote s:Startup-sequence Testnote
```

Example 5:

```
1>LIST Testfile LFORMAT "%-25N %L %12A %D
Testfile                 28     ----rwed 28-May-92
```

%-25N specifies that the length of the field containing the name of the file is to be 25 characters long (25), and the text is to be left justified (-). %12A specifies that the length of the field containing the file attributes is to be 12 characters long (12), and the text is to be right justified.

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 1:

If you have quit the Workbench and are working through a Shell, entering:

```
1> LOADWB
```

will bring the Workbench back. Entering LOADWB when the Workbench is already loaded has no effect.

Example 2:

```
1> PATH DF2:bin ADD
1> LOADWB NEWPATH
```

loads the Workbench. Any Shells started from the icon will have the same path as the Shell used to run the LOADWB NEWPATH command.

LOCALE

Format:	LOCALE [FROM <filename>] [EDIT] [USE] [SAVE] [PUBSCREEN <public screen name>]
Template:	FROM,EDIT/S,USE/S,SAVE/S,PUBSCREEN/K
Purpose:	To allow choice of languages available on system.
Path:	SYS: Prefs/Locale

Specification:

LOCALE allows you to choose the languages you want to use in your system. The default setting is USA English; however, you can choose from a number of other languages.

The Locale editor allows you to select your country of origin, your time zone, and your preferred languages. When you install your Workbench software, you choose the languages that you want available on your system.

LOCALE without any arguments or with the EDIT argument opens the Locale 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 Locale editor. For example, if you have saved a special configuration of the Locale 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.

Full information on localization is given in *Using the Amiga Workbench* Manual.

LOCK

Format:	LOCK <drive> [ON | OFF] [<passkey>]
Template:	DRIVE/A,ON/S,OFF/S,PASSKEY
Purpose:	To set the write protect status of a device.
Path:	C:LOCK

Specification:

LOCK sets or unsets the write protect status of a device 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 example, if you try to create a new directory by entering the following:

```
1> MAKEDIR WORK:Test
```

the following message will appear:

```
Can't create directory Work:Test
Disk is write-protected
```

To unlock the partition, enter:

```
1> LOCK Work: OFF SecretCode
```

The locking of a device is only good for the duration of the current session. Resetting or turning off the Amiga cancels the locking of the device.

MAGTAPE

Format:	MAGTAPE [DEVICE <device name>] [UNIT <n>] [RET I RETENSION] [REW I 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 I RETENSION option runs the tape to the end, then rewinds it. The REW I REWIND option rewinds 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 directories with the names you specify. The command works within only one directory level at a time, so any directories on the given paths must already exist. The command fails if a directory or a file of the same name already exists in the directory in which you are attempting to create a new directory.

MAKEDIR does not create a drawer icon for the new directory.

Example 1:

```
1> MAKEDIR Tests
```

creates a directory Tests in the current directory.

Example 2:

```
1> MAKEDIR DF1:Xyz
```

creates a directory Xyz in the root directory of the disk in DF1:.

Example 3:

```
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, 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.

Soft links, which can be links across volumes, are not currently implemented.

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.

MEMACS

Format:	MEMACS [<filename>] [OPT W] [goto <n>]
Template:	FROM/M,OPT/K,GOTO/K
Purpose:	To enable screen-oriented text editing.
Path:	Extras: Tools/MEmacs

Specification:

MEmacs, which stands for MicroEmacs, is a screen-oriented text editor found in the Tools drawer of the Extras disk. A text editor has the basic functionality of a word processor, but it does not support style formatting options. MEmacs is described in detail in Chapter 6.

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.

Command keys for MORE are explained on screen when you press the H key.

MORE also accepts input from a PIPE. Since standard input from a PIPE is of unknown length, the Previous Page (Backspace or b), Last Page (>), and More N% into file (%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 (press 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/M,FROM/K
Purpose:	To make a device connected to the system available.
Path:	C:MOUNT

Specification:

MOUNT causes AmigaDOS to recognize new devices, handlers, or file systems that are being added to the system. The DEVICE keyword specifies the devices to be mounted.

A MountList file contains the parameters of the device that is being mounted. By default, MOUNT looks for the MountList file in DEVS:DOSDrivers. If it does not find the file, it will then look in SYS:Storage/DOSDrivers.

Each DOSDriver file has an icon associated with it that represents the Mount file describing the device. The ability to drag DOSDriver icons means that you can control what device gets mounted in your system. This is important for CrossDOS. For example, by double clicking on the PC0 icon, you can activate MS-DOS device unit 0. By dragging the PC0 icon from Storage/DOSDrivers to Devs/DOSDrivers, you make the selection permanent, and PC0: is then available after every reboot.

Multiple devices can be mounted with just one invocation. Different actions are taken depending on whether the device name argument ends with a colon.

Example 1:

```
1> Mount PIPE:
```

This looks for the file DEVS:DOSDrivers/PIPE and processes it if found. If the Mount file is processed successfully, the Tool Types for DEVS:DOSDrivers/PIPE.info are then processed and override any settings from the Mount file.

If DEVS:DOSDrivers/PIPE does not exist, an attempt is made to find SYS:Storage/DOSDrivers/PIPE. If this also fails, then an attempt is made to find a PIPE entry in DEVS:MountList.

Example 2:

```
1> Mount PIPE
```

When there is no colon at the end of a device name argument, the name is taken as the filename of the Mount file to process. This filename can contain wildcards so something like the following is possible: Mount DEVS:DOSDrivers/~(#?.info).

Example 3:

```
1> Mount PIPE FROM <MountList>
```

This scans for a PIPE entry in <MountList>.

Any keyword that can be put in a MountList can also be put as a Tool Type entry. The Tool Types data overrides the MountList file itself.

See Appendix B for further information on MountList.

MULTIVIEW

Format: MULTIVIEW [FILE <filename>] [CLIPBOARD] [CLIPUNIT <clipboard unit>] [SCREEN] [PUBSCREEN <public screen name>] [REQUESTER] [BOOKMARK] [FONTNAME <font name>] [FONTSIZE <font size>] [BACKDROP] [WINDOW]

Template: FILE,CLIPBOARD/S,CLIPUNIT/K/N,SCREEN/S, PUBSCREEN/K,REQUESTER/S,BOOKMARK/S, FONTNAME/K,FONTSIZE/K/N,BACKDROP/S, WINDOW/S:

Purpose: To display picture files, text files, AmigaGuide help files, sound files, and animated graphics files.

Path: SYS:Utilities/MultiView

Specifications:

MULTIVIEW displays the contents of a file. If the file is not in the current directory, you must specify the complete path. If you don't specify a file, MULTIVIEW will display a file requester.

If CLIPBOARD is specified, the Clipboard will be viewed instead of the file. CLIPUNIT specifies the Clipboard unit to use when using the CLIPBOARD keyword.

SCREEN indicates that you want the object to appear on its own screen, using the display mode specified by the object. For example, if an ILBM was Low Res, then the screen would match.

If REQUESTER is specified, MULTIVIEW will display a file requester.

BOOKMARK recalls the object and position when opening a file with a bookmark.

FONTNAME specifies which font to use when viewing text objects. FONTSIZE specifies the font size in points to use when viewing text objects.

BACKDROP indicates that the window should be a backdrop window.

WINDOW will open the MultiView window without an object so that it can be placed in the WBStartup directory.

MULTIVIEW is valid on Workbench 3.0 level software.

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, path, 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.

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.)

NEWSHELL uses the default startup file S:Shell-startup, unless a FROM filename is specified. S:Shell-startup is a standard AmigaDOS script file. You might have several different Shell-startup files, each having different command aliases, for example. You can call such customized Shell environments with FROM.

The NEWCLI command has the same effect as NEWSHELL; it invokes a new Shell process.

Example 1:

```
1> NEWSHELL
```

a new Shell window will open.

Example 2:

```
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.

Example 3:

```
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<priority>]
Template:	CX_PRIORITY/N/K
Purpose:	To disable the Caps Lock key.
Path:	Extras:Tools/Commodities/NoCapsLock

Specification:

NOCAPSLOCK is a Commodity Exchange program that temporarily disables the Caps Lock key.

CX_PRIORITY 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 exit NoCapsLock once it has been started from the Shell, enter Ctrl+C or use the BREAK command.

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 pressing Ctrl+C.

Example:

```
1> NOFASTMEM
```

OVERSCAN

Format:	OVERSCAN [FROM <filename>] [EDIT] [USE] [SAVE] [PUBSCREEN <public screen name>]
Template:	FROM,EDIT/S,USE/S,SAVE/S,PUBSCREEN/K
Purpose:	To change the sizes of the display areas for text and graphics.
Path:	Extras: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 example, 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 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: Extras: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 example, 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] [REMOVE] [QUIET]
Template:	PATH/M,ADD/S,SHOW/S,RESET/S,REMOVE/S, QUIET/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 simply 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 example, 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.

Example:

```
1> PATH EXTRAS:Tools ADD
```

adds the Tools directory on the Extras disk to the search path of the Shell. If the Extras disk is not in a disk drive, a requester will ask you to insert it in any drive.

If you remove Extras from the drive, and enter:

```
1> PATH
```

a list of directories in the search path will be displayed. A requester will ask you to insert Extras. However, if you enter:

```
1> PATH QUIET
```

the list of directories in the search path will be displayed. However, when the path comes to Extras:Tools, only the volume name, Extras:, will appear in the list.

See also: ASSIGN

POINTER

Format:	POINTER [FROM <filename>] [EDIT] [USE] [SAVE] [CLIPUNIT <clipboard unit>]
Template:	FROM,EDIT/S,USE/S,SAVE/S,CLIPUNIT/K
Purpose:	To change the appearance of the screen pointer.
Path:	Extras: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 example, 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.

CLIPUNIT determines which Clipboard Unit the pointer will use during cut and paste operations.

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] [PUBSCREEN <public screen name>] [UNIT <unit>]
Template:	FROM,EDIT/S,USE/S,SAVE/S,PUBSCREEN/K, UNIT/S
Purpose:	To specify a printer and print options.
Path:	Extras: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 example, 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.

PRINTER UNIT opens the Printer editor with a Device Unit gadget added to the editor. Select the unit to which output should be sent. If PRINTER UNIT <unit> is specified, output will be sent to the specified <unit>. This option is primarily available for use with old application programs.

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] [PUBSCREEN <public screen name>]
Template:	FROM,EDIT/S,USE/S,SAVE/S,PUBSCREEN/K
Purpose:	To specify graphic printing options.
Path:	Extras: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 example, 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.

PRINTERPS

Format: PRINTERPS [FROM <filename>] [EDIT] [USE] [SAVE] [PUBSCREEN <public screen name>]

Template: FROM,EDIT/S,USE/S,SAVE/S,PUBSCREEN/K

Purpose: To control features of PostScript printers.

Path: Extras: Prefs/PrinterPS

Specification:

PRINTERPS without any arguments or with the EDIT argument opens the PrinterPS editor. The PrinterPS editor enables you to control features of PostScript printers. This editor only applies if you have a PostScript printer and if you choose PostScript in the Printer preferences editor. The PrinterPS editor combines PostScript specific options for controlling text and graphic output.

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 PrinterPS editor. For example, if you have saved a special configuration of the PrinterPS 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.

See the *Using the Amiga Workbench* manual for more information.

PRINTFILES

Format: PRINTFILES {[-f] <filename>}

Template: -f/S,FILENAME/A/M

Purpose: To send file(s) to the printer.

Path: Extras: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 prior to printing each file to ensure they each begin on a new page.

PROMPT

Format:	PROMPT [<prompt>]
Template:	PROMPT
Purpose:	To change the prompt string of the current Shell.
Path:	Internal

Specification:

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.

In the examples in this manual, the prompt string is shown as 1>.

The default prompt string is:

```
%N.%S>
```

which 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 apostrophes (`).

PROMPT alone, without a string argument, resets the prompt to the default.

Example 1:

```
1> PROMPT %N
1
```

Only the Shell number is shown. The > is removed from the prompt.

Example 2:

```
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 >.

Example 3:

```
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:

s	The file is a script.
p	The file is a pure command and can be made resident.
a	The file has been archived.
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.

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 example, a file that is readable, writeable and deletable, will have ----rw-d in the protection field.

To specify the entire protection field at once, 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 letters 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 letters. 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.

Example 1:

```
1> PROTECT DF0:Memo +rw
```

sets only the protection bits r (readable) and w (writeable) to the file Memo on DF0:. No other protection bits are changed.

Example 2:

```
1> PROTECT L:#? e SUB
```

clears the e (executable) protection bit from all the files in the L: directory.

Example 3:

```
1> PROTECT Work:Paint rwed
```

The protection status of Paint becomes "----rwed".

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."
```

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 Workbench: MyDisk
```

changes the name of the Workbench disk to MyDisk. 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 [<device>] [FORCE]
Template:	DEVICE,FORCE/S
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 DEVICE 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 device 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. 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.

The colon before the directory indicates that the directory is in the root directory.

Example 1:

```
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.

Example 2:

```
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.

REQUESTCHOICE

Format:	REQUESTCHOICE <title> <body> <gadgets> [PUBSCREEN <public screen name>]
Template:	TITLE/A,BODY/A,GADGETS/A,PUBSCREEN/K
Purpose:	To allow AmigaDOS and ARexx scripts to use the Intuition EasyRequest() feature.
Path:	C:REQUESTCHOICE

Specification:

REQUESTCHOICE allows AmigaDOS and ARexx scripts to use the Intiution EasyRequest () feature.

The <title> argument specifies the title of the requester.

The <body> argument specifies the text of the requester. Linefeeds can be embedded using *n.

The <gadgets> argument specifies the text for the different gadgets. The gadget labels are separated with |.

The number of the selected gadget is printed as a result to the console. For evaluation in a script file you can redirect this output into an environment variable. If the requester could not be opened, the command will generate a return code of 20.

REQUESTCHOICE is valid on Workbench 3.0 level software.

Examples:

```
1> RequestChoice >ENV:rcnum "New Title" "This is my
requester,*nselect a gadget" "OK|Maybe|Cancel"
```

REQUESTFILE

Format:	REQUESTFILE [DRAWER <drawer name>] [FILE <file>] [PATTERN <pattern>] [TITLE <title>] [POSITIVE <text>] [NEGATIVE <text>] [ACCEPTPATTERN <pattern>] [REJECTPATTERN <pattern>] [SAVEMODE] [MULTISELECT] [DRAWERSONLY] [NOICONS] [PUBSCREEN <public screen name>]
Template:	DRAWER,FILE/K,PATTERN/K,TITLE/K, POSITIVE/K,NEGATIVE/K,ACCEPTPATTERN/K, REJECTPATTERN/K,SAVEMODE/S, MULTISELECT/S,DRAWERSONLY/S,NOICONS/S, PUBSCREEN/K
Purpose:	To allow AmigaDOS and ARexx scripts to use the ASL file requester.
Path:	C:REQUESTFILE

Specification:

REQUESTFILE allows AmigaDOS and ARexx scripts to use the ASL file requester.

The DRAWER argument specifies the initial contents of the Drawer gadget.

The FILE option specifies the initial contents of the File gadget.

The PATTERN option allows the use of a standard AmigaDOS pattern. It specifies the initial contents of the Pattern gadget. If this option is not provided, the file requester will not have any Pattern gadget.

The TITLE option specifies the title of the requester.

The POSITIVE option specifies the text to appear in the positive (left) choice in the file requester.

The NEGATIVE option specifies the text to appear in the negative (right) choice in the file requester.

The ACCEPTPATTERN option specifies a standard AmigaDOS pattern. Only files matching this pattern will be displayed in the file requester.

The REJECTPATTERN option specifies a standard AmigaDOS pattern. Files matching this pattern will not be displayed in the file requester.

If SAVEMODE is specified, the requester will operate in save mode. If MULTISELECT is specified, the requester will allow multiple files to be selected at once. If DRAWERSONLY is specified, the requester will not have a File gadget. This effectively turns the file requester into a directory requester. If NOICONS is specified, the requester will not display icons (.info files).

The selected files are returned on the command line, enclosed in double quotes and separated with spaces. The command generates a return code of 0 if the user selected a file, or 5 if the user cancelled the requester.

REQUESTFILE is valid on Workbench 3.0 level software.

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 re-entrant and re-executable. A re-entrant 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. 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 re-entrant and re-executable. Although it is unlikely, some of your programs may be pure enough to be fully re-entrant and usable by more than one process at the same time. Other programs may not be fully re-entrant 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 example, 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.

Example 1:

```
1> RESIDENT C:COPY
```

makes the COPY command resident (replaces any previous version).

Example 2:

```
1> RESIDENT Copy2 DF1:C/COPY ADD
```

adds another version of COPY to the resident list, under the name Copy2.

Example 3:

```
1> RESIDENT Xdir DF1:C/Xdir PURE
```

makes an experimental, non-pure version of the DIR command resident.

Example 4:

```
1> RESIDENT CD REMOVE
```

makes the Internal CD command unavailable.

Example 5:

```
1> RESIDENT CD REPLACE
```

restores the CD command to the system.

See Also: PROTECT, LIST.

RUN

Format:	RUN <command> [+ {<command>}]
Template:	COMMAND/F
Purpose:	To execute commands as background processes.
Path:	Internal

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 the 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.

Example 1:

```
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.

Example 2:

```
1> RUN EXECUTE Comseq
```

executes, in the background, all the commands in the file Comseq.

SCREENMODE

Format:	SCREENMODE [FROM <filename>] [EDIT] [USE] [SAVE]
Template:	FROM,EDIT/S,USE/S,SAVE/S
Purpose:	To select a display mode.
Path:	Extras: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 example, 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] <string | pattern> [ALL] [NONUM] [QUIET] [QUICK] [FILE] [PATTERN]
Template:	FROM/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 insensitive (capitalization is ignored).

The options are:

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, press Ctrl+D. SEARCH is aborted when a Ctrl+C is pressed.

SERIAL

Format:	SERIAL [FROM <filename>] [EDIT] [USE] [SAVE] [PUBSCREEN <public screen name>] [UNIT <unit>]
Template:	FROM,EDIT/S,USE/S,SAVE/S,PUBSCREEN/K, UNIT/S
Purpose:	To set the specifications for communication through the serial port.
Path:	Extras: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 example, 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.

SERIAL UNIT opens the Serial editor with a Device Unit gadget added to the editor. Select the unit to which output should be sent. If SERIAL UNIT <unit> is specified, output will be sent to the specified <unit>.

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 lists 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 in the new Shells.

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:

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. In systems using Kickstart 2.0 or later, this is done automatically during the boot process.

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 system clock is set with the time saved in the hardware clock.

Amiga 500s do not have battery backed-up clocks, unless an A501 RAM expansion cartridge has been installed.

See also: DATE

SETDATE

Format:	SETDATE <file | pattern> [<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 ALL changes the date and time of all the files in the specified directory.

SETDATE does not affect the software or hardware clocks.

The output of the DATE command may be used as input to SETDATE.

Example 1:

```
1> SETDATE TestFile
```

changes the date and time associated with TestFile to the current date and time.

Example 2:

```
1> SETDATE TestFile 16-09-89 15:25:52
```

changes 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 comand line. This includes command lines that are part of a script.

SETENV with no arguments lists the current global variables. Global variables are stored in the ENV: directory and are used by all processes. However, 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.

Example 1:

```
1> SETENV Editor Extras: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 Extras disk. The variable Editor will be available in any Shell.

Example 2:

```
1> SETENV Editor C:ED
```

same as above, only the editor specified is ED.

```
1> ECHO $Editor
C:ED
```

See Also: GETENV, UNSETENV

SETFONT

Format:	SETFONT <font> <size> [SCALE] [PROP] [ITALIC] [BOLD] [UNDERLINE]
Template:	NAME/A,SIZE/N/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.

SETKEYBOARD

Format: SETKEYBOARD <keymap name>

Template: KEYMAP/A

Purpose: To set the keymap for the Shell.

Path: C: SETKEYBOARD

Specification:

SETKEYBOARD specifies the keymap used by the Amiga. The available files are listed below:

Keymap	Keyboard
cdn	French Canadian
ch1	Swiss French
ch2	Swiss German
d	German
dk	Danish
e	English
f	French
gb	Great Britain
i	Italian
n	Norwegian
po	Portuguese
s	Swedish
usa0	(for programs developed before V1.0)
usa	American
usa2	Dvorak

To have the system always use a keymap other than the default, add the SETKEYBOARD command to your Startup-sequence file, or use the Input editor for a permanent choice.

Example:

To change to a French Canadian keymap, enter:

```
1> SETKEYBOARD cdn
```

The keymap file must be in the KEYMAPS: directory for SETKEYBOARD to find it.

SETPATCH

Format:	SETPATCH [QUIET] [NOCACHE] [REVERSE]
Template:	QUIET/S,NOCACHE/S,REVERSE/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.

If QUIET is specified, no output will be sent to the screen.

NOCACHE will prevent Data Cache from being turned on on 68030 systems made by some third party vendors.

REVERSE allocates memory from the top. This option is primarily available for CDTV developers.

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 as far back 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."
QUIT
ENDIF
LAB message
ECHO "The <name> file does exist."
```

You can run the script by entering:

```
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 reading to the right, 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.

SOUND

Format:	SOUND [FROM <filename>] [EDIT] [USE] [SAVE] [PUBSCREEN <public screen name>]
Template:	FROM,EDIT/S USE/S SAVE/S PUBSCREEN/K
Purpose:	To control type of sound and sound attributes produced by Amiga.
Path:	Extras: Prefs/Sound

Specification:

The Sound editor allows you to determine the type of sound and its attributes that you want your Amiga to produce when the system issues an application prompt or error notification.

SOUND without any arguments or with the EDIT argument opens the Sound 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 Sound editor. For example, if you have saved a special configuration of the Sound 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.

STACK

Format:	STACK [<stack size>]
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 a 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 25,000 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.

Caution **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.

Example 1:

```
1> STATUS 1
Process 1: Loaded as command: status
```

Example 2:

```
1> STATUS 1 FULL
Process 1: stk 4000, gv 150, pri 0 Loaded as command:
status
```

Example 3:

```
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] [SAVE] [PUBSCREEN <public screen name>]
Template:	EDIT/S,SAVE/S,PUBSCREEN/K
Purpose:	To set the system clock.
Path:	Extras:Prefs/Time

Specification:

TIME without any arguments or with the EDIT argument opens the Time editor.

The SAVE option causes the current system time to be saved to the battery backed-up clock, similar to SETCLOCK SAVE.

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

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 #>]

Template: NAME,VERSION/N,REVISION/N,UNIT/N, 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 39.92 Workbench version 39.1

1> VERSION Prefs/Font
Prefs/Font version 38.21
```

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.

Example 1:

```
1> WAIT 10 MINS
```

waits ten minutes.

Example 2:

```
1> WAIT UNTIL 21:15
```

waits until 9:15 p.m.

WBPATTERN

Format:	WBPATTERN [FROM <filename>] [EDIT] [USE] [SAVE] [CLIPUNIT <clipboard unit>]
Template:	FROM,EDIT/S,USE/S,SAVE/S,CLIPUNIT/K/N
Purpose:	To create background patterns for the Workbench and windows.
Path:	Extras:Prefs/WBPattern

Specification:

WBPATTERN with no arguments or with the EDIT argument opens the WBPattern 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 WBPattern editor. For example, if you have saved a special configuration of the WBPattern 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.

CLIPUNIT specifies which Clipboard unit to use during cut and paste operations.

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 examples of the named item have been found and listed. This ensures 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:
Workbench: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.

Chapter 6
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.

Using 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 pre-programmed with the most common operations, but they can be reconfigured to suit your personal preferences. All ED functions are accessible through the keyboard.

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.

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 arrow keys allow for quick movement throughout a file.
- Scrolling speed has been increased.
- Configuration script files can be created 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 S:Ed-startup file 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 pre-programmed 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, 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 entered them at the keyboard. You can even include Quit commands. This is a way to program ED completely externally and use it as a filter in scripts.

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.

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 three.

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.

ED always appends a line feed to the end of a file, even if you do not specify one.

ED Commands

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, 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 enter appears on the command line at the bottom of the window. ED does not execute the command until you press Return. You can enter 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 6-20.

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:/

:ECHO "Hello!":

"864 bytes"
```

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. 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. Be sure to include a space before and after the question mark.

For example, Save (Esc,S,A) is an extended command that saves the contents of ED to a specified file. If you press Esc then enter:

```
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, enter:

```
SA ? !Save As?!
```

The first question mark tells the Save command 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 Mode Commands

This section describes the immediate mode commands, which control the following:

- cursor movement
- text scrolling
- text insertion
- text deletion
- repetition of commands

Moving the Cursor in Immediate Mode

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 arrow keys, Tab, and several Ctrl + key combinations.

To move the cursor one position in any direction, press the appropriate arrow 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 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 arrow	Top of the file.
Shift+down arrow	Bottom of the file.
Shift+left arrow	Left edge of the ED window (regardless of the margin setting).
Shift+right arrow	End of the current line.
Ctrl+]	Right edge of the current line (if the cursor is already there, it is moved to the left edge).
Ctrl+E	Start of the first line on the screen (if the 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 three).

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 arrow key. The text will scroll one line at a time. You can move the text in larger amounts 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 in Immediate Mode

While in immediate mode, any characters you enter 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 entering. If you move the cursor beyond the end of the line, by using the Tab or arrow 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 at the cursor position 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 in Immediate Mode

ED has no typeover mode. To replace a word or line, you must delete the existing words and insert the 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 at the cursor position.
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 in Immediate Mode

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 kept 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 will keep moving right.

Extended Mode Commands

This section describes the extended mode commands which manage:

- program control
- cursor movement
- text altering
- block control
- searching and exchanging text

To enter extended command mode, press Esc. An asterisk appears as a prompt. 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 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 example:

```
T; F /Workbench/
```

moves the cursor to the top of the file, then searches for the next occurrence of the word Workbench.

Program Control in Extended Mode

This section provides a specification of the program control commands.

New Project Esc,N,W

Creates a new file, replacing the existing file. If changes have been made to 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,O,P

Opens a file. If you want a file requester to appear, enter 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, if changes have been made to the existing file, 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,R,F

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.

Caution ED cannot undo a line deletion. Once you have moved from the current line, the Undo command cannot undo a change.

Show Esc,S,H

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.

Redisplay Esc,V,W

Refreshes the screen.

Set Tab Esc,S,T

Sets the tab stop. To change the current setting of tabs, use the Set Tab command followed by a number. For example:

```
ST 5
```

sets the tab stops to every fifth column.

Set Left Margin Esc,S,L

Sets the left margin. To specify the left margin, use the Set Left Margin 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 example, if you have an 80 column screen, you should not set the left margin to 81.

Set Right Margin Esc,S,R

Sets the right margin. To specify the right margin, use the Set Right Margin command followed by a number indicating the column position. For example:

```
SR 80
```

sets the right margin to the 80th column.

Extend Margins Esc,E,X

Extends the margins for the current line. Once you have entered the Extend Margins command, ED ignores the right margin on the current line. When you move the cursor from the current line, ED turns the normal margins on again.

Status Line Message Esc,S,M

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,S,A

Saves the text. If no filename is specified, Save 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, enter:

```
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:.

Esc,S,A followed by Q is equivalent to the Exit 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 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. It is then overwritten with the latest contents of ED. If the T: directory does not exist, ED does not make a backup.

Exit with Query Esc,X,Q

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. Exit with Query 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.

Moving the Cursor in Extended Mode

There are several commands for moving the cursor around the screen as 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,E,P	End of a page
Esc,P,D	Next page
Esc,P,U	Previous page
Esc,N	Start of next line
Esc,P	Start of previous line
Esc,C,L	One character to the left
Esc,C,R	One character to the right
Esc,C,E	End of current line
Esc,C,S	Start of current line
Esc,T,B	Next tab position
Esc,W,N	Start of next word
Esc,W,P	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 in Extended Mode

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 Insert Before 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 Insert Before, 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,D,C	Deletes the character at the cursor position
Esc,D,L	Deletes the character to the left of the cursor
Esc,D,W	Deletes to the end of the current word
Esc,E,L	Deletes to the end of the current line
Esc,F,C	Switches the case of letters

Block Control in Extended Mode

To move, insert, or delete text, use the commands described in this section.

Block Start Esc,B,S
Block End Esc,B,E

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 Block Start command. Move the cursor to the last line that you want in the block, then give the Block End command. ED selects the entire line no matter where on the line the cursor is positioned.

If you have defined a block with Block Start and Block End, 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 Insert Block 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 enter:

```
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,I,B

Inserts a copy of the block after the current line. The block will remain defined until you change the text. You can keep using Insert Block to insert copies of the block throughout the document.

Delete Block Esc,D,B

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 Delete Block, then Insert Block.) If you need to place a copy of a block elsewhere, select it, insert the block in the new location, then delete it from the original location.

Show Block Esc,S,B

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,W,B

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 entering:

```
WB ? /File?/
```

Insert File Esc,I,F

Inserts a file into the current file. ED inserts 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, placing it on the line beneath the current cursor position.

Searching and Exchanging in Extended Mode

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 Upper Case command is used (explained on page 6-19).

For example:

```
F !Workbench!
```

searches through the file for the next occurrence of the word Workbench.

Backward Find Esc,B,F

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 entering 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.

ED ignores margin settings when exchanging text.

Exchange and Query Esc,E,Q

Searches for text to be exchanged, but asks for permission to do so. When you use Exchange and Query, 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 entering:

```
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. Exchange and Query is usually given in repeated groups.

Upper Case	Esc,U,C
Lower Case	Esc,L,C

Specifies a case-insensitive search. To tell all subsequent searches not to make any distinction between upper and lowercase text, use the Upper Case command. To make searches case sensitive again, use the Lower Case command.

Repeating Commands

ED remembers any extended command line you enter. To execute a command line again, you do not have to re-enter it, you can press Ctrl+G. For example, 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 Repeat (Esc,R,P) 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 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 Run Command File (Esc,R,F) extended command.

Set Menu Item Esc,S,I

Defines the menu headings and items. There are 120 menu item slots that you can fill. The syntax for Set Menu Item 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 slot types and their functions are:

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.

Caution **Do not create a menu without items. This could cause problems with your system.**

Enable Menu Esc,E,M

Enables the menus. This command must be given after the Set Menu Item 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 two menus:

```
Project                 Move
Open ...                Top
Save ...                Bottom

Quit
```

Set Function Key Esc,S,F

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 arrow 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 example, 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.

Display Function Key Esc,D,F <key>

Displays the setting for the function key specified by <key>. For instance, using the above example, if you pressed Esc,D,F1, the status line would display T to indicate that the T command is linked to F1.

Reset Key Esc,R,K

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 arrow	Move to beginning of line
22	Shift+right arrow	Move to end of line
23	Shift+up arrow	Move to top of document
24	Shift+down arrow	Move to bottom of document
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

Slot #	Key/Key Sequence	Function
55	Ctrl+]	Move to end or start of line, depending on cursor position
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 example, 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/'
```

assigns values to the fifteen variables outlined below:

stem.LEFT	Current left margin (Set Left Margin command)
stem.RIGHT	Current right margin (Set Right Margin command)
stem.TABSTOP	Current tab stop setting (Set Tab 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 (Extend Margins command)
stem.FORCECASE	Case sensitivity flag (Upper Case/Lower Case 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

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.

You can use several extended commands from ARexx, as illustrated in the example program, Transpose.ed (below). 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, enter 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 enter:

```
RX /transpose.ed/
```

If you have entered the Transpose.ed program correctly and ARexx is set up and running properly, the program will execute and the characters will be transposed. Please note the entire filename, including the extension, must be specified in order for the program to work .

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                 */
```

Using 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 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 entering past the rightmost character on the line, but what you enter will not show on the screen.

Starting MEmacs

MEmacs is in the Tools directory of the Extras 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 Extras disk. If you have a hard disk, the Tools drawer will be in your System 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 on which the cursor will appear 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.

MEmacs Commands

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 uses to store the text you are editing.

If you specify 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 simultaneously. 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.

MEmacs has two modes of operations: normal and command. When MEmacs is in normal mode, you can:

- move the cursor using the arrow keys.
- move the cursor to the edge of the window by holding down Shift and pressing the appropriate arrow key.
- move the cursor by clicking the left mouse button at the desired place on the screen.
- insert characters at the current cursor position by entering 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 MEmacs is in the command mode, the cursor jumps to the bottom line of the display, and the program asks you for additional information. The command mode is entered through various menu items which are explained later.

There are some special terms associated with MEmacs that you should be familiar with.

Buffer	A memory area that MEmacs controls. There is always at least one buffer used by MEmacs, 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 inserted into your document by using the Yank command described on page 6-35. 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 a 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 example, the Rename menu item shows ^XF 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 example, 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, 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 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 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 a file 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:
```

Enter the complete path of the file, and press Return. MEmacs will read the file into a buffer, if it is not already in a buffer. If the file is already in a buffer, MEmacs will switch you to that buffer automatically. 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.

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 responds "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 and did not 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 enter 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, 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 inserted into the document by using the Yank command, described below.)

If a block of text has been marked using the Set-mark command (described on page 6-36) 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 location, 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 elsewhere 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 four 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. When you choose Select-buffer, MEmacs moves the cursor to the bottom line and prompts:

```
Use buffer:
```

If you specify one of the names shown in the List-buffers listing, 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 enter 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 re-use.

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 edge of all the lines in the current buffer. The text is re-arranged 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 example, 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 enter 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 arrow 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 entering 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, then execute the command.

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 example, 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 are not working in) to the next page. This doesn't make the window available for editing; it only 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 then press Return. 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 (dot) and moves the cursor to where the mark had previously 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 cursor toward the end of the buffer by one full window, less one line. The text scrolls up the screen. The cursor is repositioned so as to stay on the screen.

Prev-page Esc,V

Moves the cursor toward the beginning of the buffer by one full window, less one line. The text scrolls down the screen. 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 at the cursor position to become the first character of the following line. This command leaves the cursor in the original line so that you can enter new characters beginning at the current cursor position.

If you select Open-line by mistake, immediately pressing Del closes 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 first position on a line.

End-of-line Ctrl+E

Moves the cursor to the last position on a line. If you have entered more characters than will fit on a line, a dollar sign ($) appears at the right 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 arrow key to move the cursor, it will take as many key presses as there are unseen characters before the cursor actually begins to move.

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 down 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 example, 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 at the cursor position 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 cursor position and proceeding to the last character of the word.

Lower-word Esc,L

Changes a word to lowercase, starting at the cursor position and proceeding to the last character of the word.

Cap-word Esc,C

Changes the character at the cursor position 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 case 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 enter 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 example, 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.

Example (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) then press the minus sign (hyphen) on the keyboard.

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.

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 pressed. 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.

Table 6-1 contains the default values of the function keys when used in macro commands.

Table 6-1. 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

Key	Assignment	Key Sequence
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 entering 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 example, this is incorrect:

```
execute-line: insert file <filename>
```

You must enter:

```
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.

Keys are bound when they can be used to perform a function. For example, any key or key sequence that can be used as a shortcut for a menu item is bound to that menu item.

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.

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 entered in the command line. This command is usually used when creating or editing executable MEmacs script files.

Move to Edge of Window Shift+Arrow

By holding down Shift and an arrow 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 to save 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	Enter 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 example, to enter the ^S character shown in the example, you would have to press Ctrl+Q, Ctrl+S.

Using 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 rather 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, as described in "*Starting EDIT*" below.

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.

EDIT Commands

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.

Any lines that are inserted into the source file, 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 entered 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 need the prompt. However, if you press Return, the prompt will re-appear.

The following circumstances cause line verification:

- When you enter 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 example, 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 example, you only need to enter 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 entered in either uppercase or lowercase.

You can enter a command in three ways:

- entering the commands, then pressing Return.
- entering the final command argument, then pressing Return.
- entering 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 section.

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 example, 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
/A/	A
?Hello?	Hello
+String without final delimiter	String without final delimiter

Commands that take two string arguments use the same delimiter for both strings and do not repeat the delimiter 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 entered.)

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. 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.

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 entered:

```
E /slay//
```

"slay" would be replaced with the null string. In other words, "slay" would be deleted from the current line.

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; B cannot be used with qualifiers 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; E cannot be used with qualifiers 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 left to right). 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 qualifiers 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 qualifiers 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 entering a number in front of it. For example, the N command allows you to move forward to the next line in the source file. Entering 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 example:

```
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 enter 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 forward four lines.

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 be repeating P or by giving a number before the P command.

P	Moves to the previous line.
4P	Moves four lines back.

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 forward 3 more lines.

Find F <string>

The F command allows you to select a current line by specifying some of its content. For example:

```
F /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 B,F <string>

The B,F 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".

B,F 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 <string2> after the first occurrence of <string1>.

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 <string1> is a null string, EDIT inserts <string2> at the beginning of the line. You can use qualifiers to further restrict the context of <string1>. If <string1> 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 <string2> before the first occurrence of <string1>. 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 <string1> is a null string, EDIT inserts <string2> at the beginning of the line. You can use qualifiers to further restrict the context of <string1>. If <string1> 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 <string1> with <string2>. 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
```

If <string1> is a null string, <string2> is inserted at the beginning of the line. If <string1> 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 source 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, Z, and Return again. For example:

```
I 8
Laughing all the way
Z
```

inserts the line "Laughing all the way" 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, enter 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, enter:

```
D . *
```

Delete all lines until the specified string is found D,F<string>

The D,F 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, enter:

```
R
<replacement text>
Z
```

To replace one line, enter the line number after R. For example:

```
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 enter:

```
I
<new text>
the end
```

Show current information about EDIT S,H,D

The S,H,D (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 T,R + | -

The T,R command allows you to preserve any blanks that fall at the end of lines. By default EDIT suppresses blanks (the T,R command is turned off). To preserve any end of the line blanks in both the input and output lines, enter:

```
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, in the following:

```
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> A,P <string1> <string2>

The A,P command inserts <string2> after the first occurence of <string1>. The pointer is then positioned after <string2>.

Insert <string2> before <string1> B,P <string1> <string2>

The B,P command inserts <string2> before the first occurence of <string1>. The pointer is then positioned after <string2>.

Exchange <string1> with <string2> E,P <string1> <string2>

The E,P command replaces the first occurence of <string1> with <string2>. The pointer is then positioned after <string2>.

Delete Till After D,T,A <string>

The D,T,A command deletes all the text from the beginning of the line or the character pointer to end of the specified string.

Delete Till Before D,B,A <string>

The D,B,A command deletes all the text from the beginning of the line or the character pointer stopping just before the specified string.

Delete From After D,F,A <string>

The D,F,A command deletes all the text from just after a specified string to the end of the line.

Delete From Before D,F,B <string>

The D,F,B command 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> S,B <string>

The S,B 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 S,B to restrict the context of the string.

Split line after <string> S,A <string>

The S,A 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 S,A to restrict the context of the string.

Join two lines C,L [<string>]

The C,L (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 renumbers 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 enter the complete word REWIND; the first four letters will 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 line verification off; the lines will not be displayed on the screen. To turn verification off, enter:

```
V -
```

To turn it back on, enter:

```
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 entered, 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 T,P

The T,P (Type Previous) command displays the lines currently held in the output queue.

Type until EDIT has replaced all the lines in the output queue T,N

The T,N (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, T,N 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 T,L <n>

The T,L command is similar to the T command in that it types the specified number of lines. However, T,L 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 example, if you want to change DF0: to DF2: throughout an entire file, enter:

```
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 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 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 re-open 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.

Command	Action
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 desination 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.

Appendix A
Error Messages

This appendix 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 unnecessary windows and applications, then re-issue the command. If it still doesn't work, reboot. Memory may be sufficient, but 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.	Verify the correct command format.
115 Bad number	The program was expecting a numerical argument.	Verify the correct command format.
116 Required argument missing	Incorrect command line.	Verify the correct command format.

Error	Probable Cause	Recovery Suggestion
117 Argument after `=' missing	Incorrect command line.	Verify the correct command format.
118 Too many arguments	Incorrect command line.	Verify the correct command format.
119 Unmatched quotes	Incorrect command line.	Verify the correct command format.
120 Argument line invalid or too long	Your command line is incorrect or contains too many arguments.	Verify 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. 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.	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 re-issue the command.

Error	Probable Cause	Recovery Suggestion
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). Re-issue 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). Re-issue 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.	Re-issue 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.	Re-type the name, being sure not to use any invalid characters or exceed the maximum length.

Error	Probable Cause	Recovery Suggestion
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.	Verify 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.
214 Disk is write-protected	The plastic tab is in the write-protect position.	Remove the disk, move the tab, and re-insert the disk, or 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.

Error	Probable Cause	Recovery Suggestion
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, or the name is mispelled.	Insert the correct floppy disk, mount the device, check the spelling of the device name, 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.

Error	Probable Cause	Recovery Suggestion
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 a 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\Read\Link to resolve the soft link and retries the operation.

Appendix B
Additional Workbench Directories

In addition to the AmigaDOS commands explained in Chapter 5, there are many other files and directories on your Workbench disk. This section discusses the S:, DEVS:, L:, FONTS:, and LIBS: directories. Some of the files are new, while others have been revised since the previous release. 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.

Except for FONTS:, 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 example, 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, the S: directory also contains the scripts 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.

SPat, DPat

These scripts add pattern matching to commands which do not 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 enter:

```
1> SPat More S:s#?
```

A script similar to this is generated:

```
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
```

changes the current directory to RAM:, then returns you to the starting directory.

The DEVS: Drawer

The DEVS: drawer 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:.
parallel.device	Controls access to the parallel port.
printer.device	Controls access to the printer device.
serial.device	Controls access to the serial port.

For more information on the .device files, see the ROM Kernel manuals published by Addison-Wesley.

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 hard disks or external disk drives, you must make the Amiga aware of the existence of the device. Some devices automount using the expansion directory. Other devices have their own mountlist files with icons in the DOSDrivers drawer in DEVS:. They get mounted automatically during the standard Startup-sequence. 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.

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 the device.

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 the following pages.

Table B-1. 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.

Keyword	Function
Reserved=	The number of blocks reserved for the boot block; should be 2.
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).
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, omit it altogether.
Baud=	Serial device baud rate.
Control=	Serial device word length, parity, and stop bits.

Keyword	Function
Forceload=	Forces a file system to be loaded from disk even though a suitable entry is in the resource list. FORCELOAD=0; /* default, check the resource list before the disk */ FORCELOAD=1 /*always load from disk*/

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 drawer within DEVS: (Devs/Keymaps). International keymaps are available in the Storage/Keymaps directory of the Extras disk. If you have a hard disk, they are in the DEVS:Keymaps drawer.

Table B-2. Available Keymaps

Keymap	Keyboard it Represents
cdn	French-Canadian
ch1	Swiss-French
ch2	Swiss-German
d	German
dk	Danish
e	Spanish
f	French
gb	British
i	Italian
n	Norwegian

Keymap	Keyboard it Represents
po	Portuguese
s	Swedish
usa0	For V1.0/ programs
usa2	Dvorak

To use an international keymap:

1. Drag the appropriate icon from the Storage/Keymaps drawer to the DEVS/Keymaps drawer. Then use the Input preferences to select it.
2. Use the Input preferences editor to inform the system of a change.

If you want to use a different keymap on a regular basis, be sure you have copied it into the DEVS/Keymap drawer.

Printers

If you have a floppy disk system, the Printers drawer of the Workbench disk is empty except for the generic driver. In order to use a printer with your Amiga, you must drag the appropriate printer driver icon from the Extras:Storage/Printers drawer to your Workbench:Devs/Printers drawer.

If you have a hard disk system, all the available printer drivers will be in your DEVS/Printers drawer.

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.

Handlers must be named in the DOSDrivers or MountList file 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-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 DOSDrivers entry is:

```
Handler = L:Aux-handler
Stacksize = 1000
Priority = 5
```

You can use Aux-Handler to use another terminal with your computer. For example:

```
1> NEWSHELL AUX:
```

Queue-Handler

The Queue-Handler is an I/O mechanism used to provide input/output communication between programs. It essentially creates an interprocess communication channel named PIPE. 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. You must provide a name after PIPE:, such as a filename.

TheDOSDrivers entry is:

```
Handler = L:Queue-handler
Stacksize = 3000
Priority = 5
GlobVec = -1
```

PIPE: is mounted by default during the standard Startup-sequence.

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.

FONTS

The FONTS: assignment refers to a disk or directory which 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 example, 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, respectively. 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 FONTS:, it is built into ROM. Even if you deleted the entire contents of FONTS:, 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.

Table B-3. LIBS: Files

.library File	Function
asl.library	File, font, and screen mode 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

.library File	Function
mathtrans.library	FFP transcendental function math routine modules
rexxsupport.library	Modules used by ARexx
rexxsyslib.library	Main ARexx modules
bullet.library	Library modules for finding and loading outline fonts.
version.library	Contains current software version and revision information.

Appendix C
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 example, 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 later in this appendix.

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 example,

```
1> LIST C:COPY
Directory "Sys:C" on Monday 15-Jun-92
copy        5496 --p-rwed 03-Jun-92 17:22:02
```

The size of the file is shown to the right of the filename. In this case, the COPY command is 5496 bytes. If it is made resident, it will consume approximately 5496 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 5.

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 LIBS: DF0:Libs PATH
ASSIGN DEVS: DF0:Devs PATH
ASSIGN C: DF0:C PATH
ASSIGN L: DF0:L 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 5.

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.

Caution **The original, unchanged Workbench disk should always be kept safe in case you need to restore a file from the original Workbench disk.**

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 requestor 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 Workbench 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 Workbench disk.

Caution **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.

Finally, if you absolutely must have more space, you can delete everything in REXXC, in addition to System/RexxMast, System/RexxMast.info, LIBS:rexxsyslib.library, and LIBS:rexxsupport.library. This will make almost 50K of disk space available. Again, this is not recommended as many application programs may call upon these files.

Note Files that you should not delete under any circumstances are listed below.

- C:IPrefs
- DEVS:parallel.device
- DEVS: MountList
- 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:, 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.

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.

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 example, you have a directory called Brushes which contains two dozen IFF files, and you want 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 copy the IFF files into RAM:, you can run Paint, load each brush directly from RAM:, change it, then save it back to RAM:. After you were finished with the files, you can 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 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 can 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 can 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. To activate a recoverable Ram Disk, you simply double-click on the RAD icon in the Storage/DOSDrivers drawer, or, if you always want it, drag the icon to the DEVS/DOSDrivers drawer.

Unlike RAM:, the size of RAD: is not dynamic. It is fixed in the RAD: DOSDrivers file. 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 Workbench disk.

You can also set up multiple RAD: devices of different sizes by copying the RAD: MountList entry and changing the name and unit number.

Index

AMIGA

Commodore

368763-01