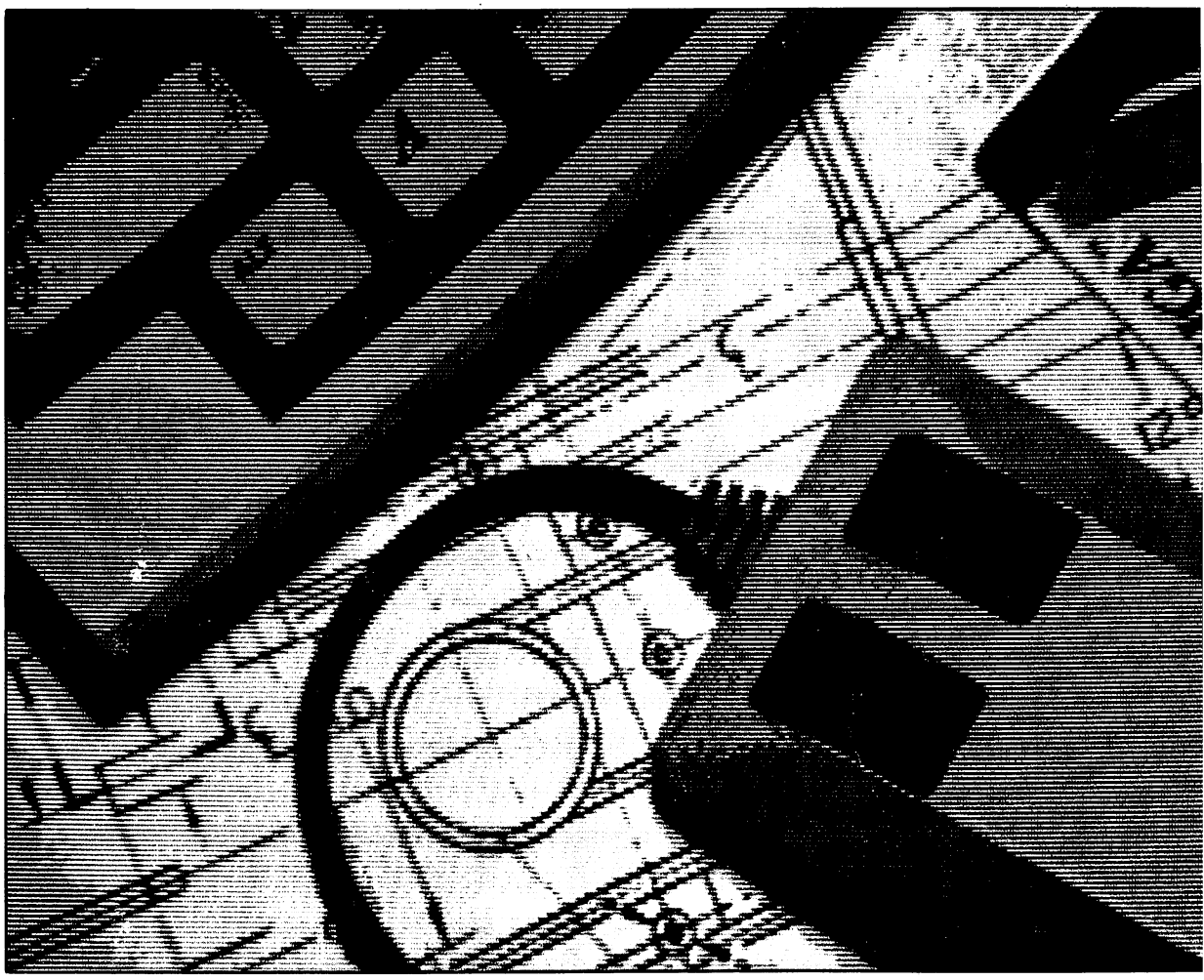
AMIGA™



AmigaDOS User's Manual

AmigaDOS User's Manual

Acknowledgements

This manual was originally written by Tim King and then completely revised by Jessica King.

A special thanks to Patria Brown whose editorial suggestions substantially contributed to the quality of the manual.

Also thanks to Bruce Barrett, Keith Stobie, Robert Peck and all the others at Commodore-Amiga who carefully checked the contents; to Tim King, Paul Floyd, and Alan Cosslett who did the same at Metacomco; and to Pamela Clare and Liz Laban who spent many hours carefully proof-reading each version.

This manual refers to Release 1, August 1985.

Printed in the U.S.A.

CBM Product Number 327264-01 rev 1.0 8.27.85

INTRODUCTION

This manual describes the various AmigaDOS, and its commands. The Command Line Interpreter (CLI) reads AmigaDOS commands typed into a CLI window and translates them into actions performed by the computer. In this sense, the CLI is similar to more "traditional" computer interfaces: you type in commands and the interface displays text in return.

Because the Workbench interface is sufficient and friendly for most users the Workbench diskettes are shipped with the CLI interface "disabled". To use the commands in this manual you must "enable" the CLI interface. This puts a new icon, labeled "cli" on your Workbench. When you have selected and opened this icon, a CLI window becomes available, and you can use it to issue text commands directly to AmigaDOS.

How To Enable the Command Line Interface

Boot your computer using the Kickstart and Workbench diskettes. Open the diskette icon. Open the "Preferences" tool. Near the left hand side of the screen, about two-thirds of the way down you will notice "CLI" with a button for "ON" and a button "OFF". Select the "ON" button. Select either "Save" or "Use" (lower right part of the Preferences screen) to leave Preferences.

How to Open a CLI Window

To use the CLI commands, you open a CLI window. Open the "System" drawer. The CLI icon (a cube containing "1>") should now be visible. Open it.

Using The CLI

To use the CLI interface select the CLI window and type the desired CLI commands. The CLI window(s) may be sized and moved just like most others. To close the CLI window, type "ENDCLI".

WORKBENCH and CLI, Their Relationship and Differences

Type "DIR" to display a list of files (and directories) in the current disk directory. This is a list of files that makes up your workbench. You may notice that there are more files in this directory than there are icons on the Workbench. Workbench only displays file "X" if that file has an associated "X.info" file. Workbench uses the ".info" file to manipulate the icon.

For example, the diskcopy program has two files. The file "DiskCopy contains the program and "Diskcopy.info" contains the Workbench information about it. In the case of painting data files like "mount.pic" the file "mount.pic.info" contains icon information and the name of the program (default) that should process it (GraphiCraft). In this case, when the user "opens" the data file (mount.pic) Workbench runs the program and passes the data file name (mount.pic) to it.

AmigaDOS sub-directories correspond to Workbench drawers. Random access block devices such as disks (DF0:) correspond to the diskette icons you have seen.

Not all programs or commands can be run under both Workbench and the CLI environment. None of the CLI commands described in chapter 2 of this manual can be run from workbench. For example, there are two separate Diskcopy commands. The one in the :c/ directory is run from AmigaDOS (CLI). The one in the system directory (drawer) is run from Workbench.

Table of Contents

Chapter 1: Introducing AmigaDOS

This chapter provides a general overview of the AmigaDOS operating system, including descriptions of terminal handling, the directory structure, and command use. At the end of the chapter, you'll find a simple example session with AmigaDOS.

Table of Contents

Section 1.3.6.)

If you find that strange characters appear on the screen when you type anything on the keyboard, you have probably pressed CTRL-O by mistake. AmigaDOS recognizes this control combination as an instruction to the console device (CON:) to display the alternative character set. To undo this condition, you press CTRL-N. Any further characters should then appear as normal. On the other hand, you could press ESC-C to clear the screen and display normal text.

Note: Any input through the console device CON: ignores function keys and cursor keys. If you want to receive these keys, you should use RAW:. (For a description.index RAW: of the console devices, see Section 1.3.6, "Understanding Device Names," later in this chapter.)

Finally, AmigaDOS recognizes all commands and **arguments** typed in either upper or lower case. AmigaDOS displays a **filename** with the characters in the case used when it was created, but finds the file no matter what combination of cases you use to specify the filename.

1.3. Using the Filing System

This section describes the AmigaDOS filing system. In particular, it explains how to name, organize, and recall your files.

1.3.1 Naming Files

AmigaDOS holds information on disks in a number of **files**, named so that you can identify and recall them. The filing system allows filenames to have up to thirty characters, where the characters may be any printing character except slash (/) and colon (:). This means that you can include space ( ), equals (=), plus (+), and double quote ("), all special characters recognized by the CLI, within a filename. However, if you use these special characters, you must enclose the entire filename with double quotes. To introduce a double quote character within a filename, you must type an asterisk (*) immediately before that character. In addition, to introduce an asterisk, you must type another asterisk. This means that a file named

```
A*B = C"
```

should be typed as follows:

```
"A**B = C*""
```

in order for the CLI to accept it.

Note: This use of the asterisk is in contrast to many other operating system s where it is used as a universal **wild card**. An asterisk by itself in AmigaDOS, represents the keyboard and the current window. For example,

```
COPY filename to *
```

copies the filename to the screen.

Avoid spaces before or after filenames because they may cause confusion.

1.3.2 Using Directories

The filing system also allows the use of **directories** as a way to group files together into logical units. For example, you may use two different directories to separate program source from program documentation, or to keep files belonging to one person distinct from those belonging to another.

Each file on a disk must belong to a directory. An empty disk contains one directory, called the **root directory**. If you create a file on an empty disk, then that file belongs to this root directory. However, directories may themselves contain further directories. Each directory may therefore contain files, or yet more directories, or a mixture of both. Any filename is unique only within the directory it belongs to, so that the file 'fred' in the directory 'bill' is a completely different file from the one called 'fred' in the directory 'mary'.

This filing structure means that two people sharing a disk do not have to worry about accidentally overwriting files created by someone else, as long as they always create files in their own directories.

WARNING: When you create a file with a filename that already exists, AmigaDOS deletes the previous contents of that file. No message to that effect appears on the the screen.

You can also use this directory structure to organize information on the disk, keeping different sorts of files in different directories.

An example might help to clarify this. Consider a disk that contains two directories, called 'bill' and 'mary'. The directory 'bill' contains two files, called 'text' and 'letter'. The directory 'mary' contains a file called 'data' and two directories called 'letter' and 'invoice'. These sub-directories each contain a file called 'jun18'. Figure 1-A represents this structure as follows:

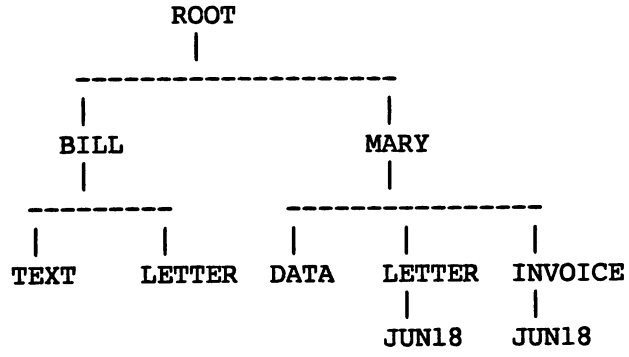


Figure 1-A: Using Directory Structure

Note: The directory 'bill' has a file called 'letter,' while the directory 'mary' contains a directory called 'letter'. However, there is no confusion here because both files are in different directories. There is no limit to the depth that you can "nest" directories.

To specify a file fully, you must include the directory that owns it, the directory owning that directory, and so on. To specify a file, you give the names of all the directories on the path to the desired file. To separate each directory name from the next directory or file name, you type a following slash (/). Thus, the full specification of the data files on the disk shown in Figure 1-A above is as follows:

```
bill/text
bill/letter
mary/data
mary/letter/jun18
mary/invoice/jun18
```

1.3.3 Setting the Current Directory

A full file description can get extremely cumbersome to type, so the filing system maintains the idea of a **current directory**. The filing system searches for files in this current directory. To specify the current directory, you use the CD (Current Directory) command. If you have set 'mary' as your current directory, then the following names would be sufficient to specify the files in that directory:

```
data
letter/jun18
invoice/jun18
```

You can set any directory as the current directory. To specify any files within that directory, simply type the name of the file. To specify files within sub-directories, you need to type the names of the directories on the path from the current directory specified.

All the files on the disk are still available even though you've set up a current directory. To instruct AmigaDOS to search through the directories from the root directory, you type a colon (:) at the beginning of the file description. Thus, when your file description has the current directory set to 'mary', you can also obtain the file 'data' by typing the description ':mary/data'. Using the current directory method simply saves typing, because all you have to do is specify the filename 'data'.

To obtain the other files on the disk, you could specify ':bill/text' and ':bill/letter' respectively. Another way might be to CD or type / before a filename. Slash does not mean "root" as in some systems, but refers to the directory above the current directory. AmigaDOS allows multiple slashes. Each slash refers to the level above. So a Unix (TM) . . / is a / in AmigaDOS. Similarly, an MS-DOS (TM) . . \ is a / in AmigaDOS. Thus, if the current directory is ':mary/letter', you may specify the file ':mary/invoice/jun18' as '/invoice/jun18'. To refer to the files in ':bill', you could type

```
CD :bill
```

or

```
CD //bill
```

Then you could specify any file in 'bill' with a single filename. Of course, you could always use the // feature to refer directly to a specific file. For example,

```
TYPE //bill/letter
```

displays the file without your first setting 'bill' as the current directory. To go straight to the root level, always type a colon (:) followed by a directory name. If you use slashes, you must know the exact number of levels back desired.

1.3.4 Setting the Current Device

Finally, you may have many disk drives available. Each disk device has a name, in the form DFn (for example, DF1), where the 'n' refers to the number of the device. (Currently, AmigaDOS accepts the device names DF0 to DF3). Each individual disk is also associated with a unique name.

In addition, the logical device SYS: is assigned to the disk you started the system up from. You can use this name in place of a disk device name (like DF0:).

The current directory is also associated with a **current drive**, the drive where you may find the directory. As you know, prefacing a file description with a colon serves to identify the root directory of the current drive. However, to give the root directory of a specific drive, you precede the colon with the drive name. Thus, you have yet another way of specifying the file 'data' in directory 'mary', that is 'DF1:mary/data'. This assumes that you have inserted the disk into drive DF1. So, to reference a file on the drive DF0 called 'project-report' in directory 'peter', you would type 'DF0:peter/project-report', no matter which directory you had set as the current one.

Note: When you refer to a disk drive or any other device, on its own or with a directory name, you should always type the colon, for example, DF1:.

Figure 1-B illustrates the structure of a file description. Figure 1-C gives some examples of valid file descriptions.

Left of the :	Right of the :	Right of a /
Device name	Directory name	Subdirectory name
-or-	-or-	-or-
Volume name	Filename	Filename

Figure 1-B: The Structure of a File Description

```
SYS:commands
DF0:bill
DF1:mary/letter
DF2:mary/letter/jun18
DOC:report/section1/figures
FONTS:silly-font
C:cls
```

Figure 1-C: Examples of File Descriptions

To gain access to a file on a particular disk, you can type its unique name, which is known as the disk's **volume name**, instead of the device name. For instance, if the file is on the disk 'MCC', you can specify the same file by typing the name 'MCC:peter/project-report'. You can use the volume name to refer to a disk regardless of the drive it is in. You assign a volume name to a disk when you format it (for further details, see "FORMAT" in Chapter 2,"Commands," later in this manual).

A device name, unlike a volume name, is not really part of the name. For example, AmigaDOS can read a file you created on DF0: from another drive, such as DF1:, if you place the disk in that drive, assuming of course that the drives are interchangeable. That is, if you create a file called 'bill' on a disk in drive DF0:, the file is known as 'DF0:bill'. If you then move the disk to drive DF1:, AmigaDOS can

still read the file, which is then known as 'DF1:bill'.

1.3.5 Attaching a Filenote

Although a filename can give some information about its contents, it is often necessary to look in the file itself to find out more. AmigaDOS provides a simple solution to this problem. You can use the command called FILENOTE to attach an associated comment. You can make up a comment of up to 80 characters (you must enclose comments containing spaces in double quotes). Anything can be put in a file comment: the day of the file's creation, whether or not a bug has been fixed, the version number of a program, and anything else that may help to identify it.

You must associate a comment with a particular file - not all files have them. To attach comments, you use the FILENOTE command. If you create a new file, it will not have a comment. Even if the new file is a copy of a file that has a comment, the comment is not copied to the new file. However, any comment attached to a file which is overwritten is retained. To write a program to copy a file and its comment, you'll have to do some extra work to copy the comment. For details, see Chapter 2 of the *AmigaDOS Developer's Manual*.

When you rename a file, the comment associated with it doesn't change. The RENAME command only changes the name of a file. The file's contents and comment remain the same regardless of the name change. For more details, see LIST and FILENOTE in Chapter 2 of this manual.

1.3.6 Understanding Device Names

Devices have names so that you can refer to them by name. Disk names such as DF0: are examples of **device names**. Note that you may refer to device names, like filenames, using either upper or lower case. For disks, you follow the device name by a filename because AmigaDOS supports files on these devices. Furthermore, the filename can include directories because AmigaDOS also supports directories.

You can also create files in memory with the device called RAM:. RAM: implements a filing system in memory that supports any of the normal filing system commands.

Note: RAM: requires the library l/ram-handler to be on the disk.

Once the device RAM: exists, you can, for instance, create a directory to copy all the commands into memory. To do this, type the following commands:

```
MAKEDIR ram:c
COPY sys:c TO ram:c
ASSIGN C: RAM:C
```

You could then look at the output with DIR RAM:. It would include the directory 'c' (DIR lists this as c(dir)). This would make loading commands very quick but would leave little room in memory for anything else. Any files in the RAM: device are lost when you reset the machine.

AmigaDOS also provides a number of other devices that you can use instead of a reference to a disk file. The following paragraphs describe these devices including NIL:, SER:, PAR:, PRT:, CON:, and RAW:. In particular, the device NIL: is a dummy device. AmigaDOS simply throws away output written to NIL:. While reading from NIL:, AmigaDOS gives an immediate 'end of file' indication. For example, you would type the following

```
EDIT abc TO nil:
```

to use the editor to browse through a file, while AmigaDOS throws away the edited output.

You use the device called SER: to refer to any device connected to the serial line (often a printer). Thus, you would type the following command sequence:

```
COPY xyz TO ser:
```

to instruct AmigaDOS to send the contents of the file 'xyz' down the serial line. Note that the serial device only copies in multiples of 400 bytes at a time. Copying with SER: can therefore appear granular.

The device PAR: refers to the parallel port in the same way.

AmigaDOS also provides the device PRT: (for PRinTer). PRT: is the printer you chose in the 'preferences' program. In this program, you can define your printer to be connected through either the serial or parallel port. Thus, the command sequence

```
COPY xyz TO PRT:
```

prints the file 'xyz', no matter how the printer is connected.

PRT: translates every linefeed character in a file to carriage return plus linefeed. Some printers, however, require files without translation. To send a file with the linefeeds as just linefeeds, you use PRT:RAW instead of PRT:.

AmigaDOS supports multiple windows. To make a new window, you can specify the device CON:. The format for CON: is as follows:

```
CON:x/y/width/height/[title]
```

where 'x' and 'y' are coordinates, 'width' and 'height' are integers describing the width and height of the new window, and 'title', which is optional, is a string. The title appears on the window's title bar. You must include all the slashes (/), including the last one. Your title can include up to thirty characters (including spaces). If the title has spaces, you must enclose the whole description in double quotes (") as shown in the following example:

```
"CON:20/10/300/100/my window"
```

There is another window device called RAW:, but it is of little use to the general user. (See Chapter 2 of the *AmigaDOS Developer's Manual* for further details.) You can use RAW: to create a raw window device similar to CON:. However, unlike CON:, RAW: does no character translation and does not allow you to change the contents of a line. That is to say, RAW: accepts input and returns output in exactly the same form that it was originally typed. This means characters are sent to a program immediately without letting you erase anything with the BACKSPACE key. You usually use RAW: from a program where you might want to do input and output without character translation.

WARNING: RAW: is intended for the advanced user. Do not use RAW: experimentally.

There is one special name, which is * (asterisk). You use this to refer to the current window, both for input or for output. You can use the COPY command to copy from one file to another. Using *, you can copy from the current window to another window, for example,

```
COPY * TO CON:20/20/350/150/
```

from the current window to the current window, for example,

```
COPY * TO *
```

or from a file to the current window, for example,

```
COPY bill/letter TO *
```

AmigaDOS finishes copying when it comes to the end of the file. To tell AmigaDOS to stop copying from *, you must give the CTRL-\ combination. Note that * is NOT the universal wild card.

1.3.7 Using Directory Conventions and Logical Devices

In addition to the aforementioned physical devices, AmigaDOS supports a variety of useful **logical devices**. AmigaDOS uses these devices to find the files that your programs require from time to time. (So that your programs can refer to a standard device name regardless of where the file actually is.) All of these 'logical devices' may be reassigned by you to reference any directory.

The logical devices described in this section are as follows:

Name	Description	Directory
SYS:	System disk root directory	:
C:	Command library	:C
L:	Library directory	:L
S:	Sequence library	:S
LIBS:	Library for Open Library calls	:LIBS
DEVS:	Device for Open Device calls	:DEVS
FONTS:	Loadable fonts for Open Fonts	:FONTS
	Temporary workspace	:T

Figure 1-D: Logical Devices

Logical device name: `SYS:`

Typical directory name: `My.Boot.Disk:`

'SYS' represents the SYStem disk root directory. When you first start up the Amiga system, AmigaDOS assigns SYS: to the root directory name of the disk in DF0:. If, for instance, the disk in drive DF0: has the volume name My.Boot.Disk, then AmigaDOS assigns SYS: to My.Boot.Disk:. After this assignment, any programs that refer to SYS: use that disk's root directory.

Logical device name: `C:`

Typical directory name: `My.Boot.Disk:c`

'C' represents the Commands directory. When you type a command to the CLI (DIR <cr>, for example), AmigaDOS first searches for that command in your current directory. If the system cannot find the command in the current directory, it then looks for 'C:DIR'. So that, if you have assigned 'C:' to another directory (for example, 'Boot_disk:c'), AmigaDOS reads and executes from 'Boot_disk:c/DIR'.

Logical device name: `L:`

Typical directory name: `My.Boot.Disk:l`

'L' represents the Library directory. This directory keeps the overlays for large commands and non-resident parts of the operating system. For instance, the disk based run-time libraries (Ram-Handler, Port-Handler, Disk-Validator, and so forth) are kept here. AmigaDOS requires this directory to operate.

Logical device name: `S:`

Typical directory name: `My.Boot.Disk:s`

'S' represents the Sequence library. Sequence files contain command sequences that the EXECUTE command searches for and uses. EXECUTE first looks for the sequence (or batch) file in your current directory. If EXECUTE cannot find it there, it looks in the directory that you have assigned S: to.

Logical device name: `LIBS:`

Typical directory name: `My.Boot.Disk:LIBS`

Open Library function calls look here for the library if it is not already loaded in memory.

Logical device name: `DEVS:`

Typical directory name: `My.Boot.Disk:DEVS`

Open Device calls look here for the device if it is not already loaded in memory.

Logical device name: `FONTS:`

Typical directory name: `My.Boot.Disk:FONTS`

Open Fonts look here for your loadable fonts if they are not already loaded in memory.

Note: In addition to the above assignable directories, many programs open files in the ':T' directory. As you recall, you find file (or directory) names predicated with a ':' in the root directory. Therefore ':T' is the directory T, within the root, on the current disk. You use this directory to store temporary files.

Programs such as editors place their temporary work files, or backup copies of the last file edited, in this directory. If you run out of space on a disk, this is one of the first places you should look for files that are no longer needed.

When the system is first booted, AmigaDOS initially assigns C: to the :C directory. This means that if you boot with a disk that you had formatted by issuing the command:

```
FORMAT DRIVE DF0: NAME "My.Boot.Disk"
```

SYS: is assigned to 'My.Boot.Disk'. The 'logical device' C: is assigned to the C directory on the same disk (that is, My.Boot.Disk:c). Likewise, the following assignments are made

```
C:          My.Boot.Disk:c
L:          My.Boot.Disk:l
S:          My.Boot.Disk:s
LIBS:       My.Boot.Disk:libs
DEVS:       My.Boot.Disk:devs
FONTS:      My.Boot.Disk:fonts
```

If a directory is not present, the corresponding logical device is assigned to the root directory.

If you are so lucky as to have a hard disk (called DH0:) and you want to use the system files on it, you must issue the following commands to the system:

```
ASSIGN SYS:         DH0:
ASSIGN C:           DH0:C
ASSIGN L:           DH0:L
ASSIGN S:           DH0:S
ASSIGN LIBS:        DH0:LIBS
ASSIGN DEVS:        DH0:DEVS
ASSIGN FONTS:       DH0:FONTS
```

Please keep in mind that assignments are global to all CLI processes. Changing an assignment within one window changes it for all windows.

If you want to use your own special font library, type

```
ASSIGN FONTS: "Special font disk:myfonts"
```

If you want your commands to load faster (and you have memory 'to burn'), type

```
makedir ram:c
copy sys:c ram:c all
assign c: ram:c
```

This copies all of the normal AmigaDOS commands to the RAM disk and reassigns the commands directory so that the system finds them there.

1.4 Using AmigaDOS Commands

To execute an AmigaDOS command, you type the name of the command, and you sometimes follow it with some further information about the information the command is to work with.

When you type a command name, the command runs as part of the Command Line Interface (CLI). You can type other command names ahead, but AmigaDOS does not execute them until the current command has finished. When a command has finished, the current CLI prompt appears. In this case, the command is running interactively.

WARNING: If you run a command interactively and it fails, AmigaDOS continues to execute the next command you typed anyway. Therefore, it can be dangerous to type many commands ahead. For example, if you type

```
COPY a TO b
DELETE a
```

and the COPY command fails (perhaps because the disk is full), then DELETE executes and you lose your file.

The CLI prompt is initially n>, where n is the number of the CLI process. However, it can be changed to something else with the PROMPT command. (For further details on the PROMPT command, see Chapter 2 of this manual.)

1.4.1 Running Commands in the Background

You can instruct AmigaDOS to run a command, or commands, in the background. To do this, you use the RUN command. This creates a new CLI as a separate process of lower priority. In this case, AmigaDOS executes subsequent command lines at the same time as those that have been RUN. For example, you can examine the contents of your directory at the same time as sending a copy of your text file to the printer. To do this, type

```
RUN TYPE text_file TO PRT:
LIST
```

RUN creates a new CLI and carries out your printing while you list your directory files on your original CLI window.

You can ask AmigaDOS to carry out several commands using RUN. RUN takes each command and carries it out in the given order. The line containing commands after RUN is called a command line. To terminate the command line, press RETURN. To extend your command line over several lines, type a plus sign (+) before pressing RETURN on every line except the last. For example,

```
RUN JOIN text_file1 text_file2 AS text_file +
SORT text_file TO sorted_text +
TYPE sorted_text to PRT:
```

1.4.2 Executing Command Files

You can also use the EXECUTE command to execute command lines in a file instead of typing them in directly. The CLI reads the sequence of commands from the file until it finds an error or the end of the file. If it finds an error, AmigaDOS does not execute subsequent commands on the RUN line or in the file used by EXECUTE, unless you have used the FAILAT command. See Chapter 2 of this manual for details on the FAILAT command. The CLI only gives prompts after executing commands that have run interactively.

1.4.3 Directing Command Input and Output

AmigaDOS provides a way for you to redirect standard input and output. You use the > and < symbols as commands. When you type a command, AmigaDOS usually displays the output from that command on the screen. To tell AmigaDOS to send the output to a file, you can use the > command. To tell AmigaDOS to accept the input to a program from a specified file rather than from the keyboard, you use the < command. The < and > commands act like traffic cops who direct the flow of information. For example, to direct the output from the DATE command and write it to the file named 'text_file', you would type the following command line:

```
DATE > text_file
```

See Chapter 2 of this manual for a full specification of the < and > symbols.

1.4.4 Interrupting AmigaDOS

AmigaDOS allows you to indicate four levels of attention interrupt with CTRL-C, CTRL-D, CTRL-E, and CTRL-F. To stop the current command from whatever it was doing, press CTRL-C. In some cases, such as EDIT, pressing CTRL-C instructs the command to stop what it was doing and then to return to reading more EDIT commands. To tell the CLI to stop a command sequence initiated by the EXECUTE command as soon as the current command being executed finishes, press CTRL-D. CTRL-E and CTRL-F are only used by certain commands in special cases. See the *AmigaDOS Developer's Manual* for details.

Note: It is the programmer's reponsibility to detect and repond with these interruption flags. AmigaDOS will not kill a program by itself.

1.4.5 Understanding Command Formats

This section explains the standard format or argument template used by all AmigaDOS commands to specify their arguments. Chapter 2 of this manual includes this argument template in the documentation of each of the commands. The template provides you with a great deal of flexibility in the order and form of the syntax of your commands.

The argument template specifies a list of **keywords** that you may use as synonyms, so that you type the alternatives after the keyword, and separate them with an =.

For example,

ABC,WWW,XYZ=ZZZ

specifies keywords ABC, WWW, and XYZ. The user may use keyword ZZZ as an alternative to the keyword XYZ.

These keywords specify the number and form of the arguments that the program expects. The arguments may be optional or required. If you give the arguments, you may specify them in one of two ways:

By position In this case, you provide the arguments in the same order as the keyword list indicates.

By keyword In this case, the order does not matter, and you precede each argument with the relevant keyword.

For example, if the command MYCOMMAND read from one file and wrote to another, the argument template would be

FROM,TO

You could use the command specifying the arguments by position

```
MYCOMMAND input-file output-file
```

or using the keywords:

```
MYCOMMAND FROM input-file TO output-file
MYCOMMAND TO output-file FROM input-file
```

You could also combine the positional and keyword argument specifications, for example, with the following:

```
MYCOMMAND input-file TO output-file
```

where you give the FROM argument by position, and the TO argument by keyword. Note that the following form is incorrect

```
MYCOMMAND output-file FROM input-file
```

because the command assumes that 'output-file' is the first positional argument (that is, the FROM file).

If the argument is not a single word (that is, surrounded or 'delimited' by spaces), then you must enclose it with quotation marks ("). If the argument has the same value as one of the keywords, you must also enclose it with quotation marks. For example, the following:

```
MYCOMMAND "file name" TO "from"
```

supplies the text 'file name' as the FROM argument, and the file name 'from' as the TO argument.

The keywords in these argument lists have certain qualifiers associated with them. These qualifiers are represented by a slash (/) and a specific letter. The meanings of the qualifiers are as follows:

/A The argument is required and may not be omitted.

/K The argument must be given with the keyword and may not be used positionally.

/S The keyword is a switch (that is, a toggle) and takes no argument.

The qualifiers A and K may be combined, so that the template

DRIVE/A/K

means that you must give the argument and keyword DRIVE.

In some cases, no keywords may be given. For example, the command DELETE simply takes a number of files for AmigaDOS to delete. In this case, you simply omit the keyword value, but the commas normally used to separate the keywords remain in the template. Thus, the template for DELETE, that can take up to ten filenames, is

,,,,,,,,,

Finally, consider the command TYPE. The argument template is

FROM/A,TO,OPT/K

which means that you may give the first argument by position or by keyword, but that first argument is required. The second argument (TO) is optional, and you may omit the keyword. The OPT argument is optional, but if it is given, you must provide the keyword. So, the following are all valid forms of the TYPE command:

```
TYPE filename
TYPE FROM filename
TYPE filename TO output-file
TYPE filename output-file
TYPE TO outputfile FROM filename OPT n
TYPE filename OPT n
TYPE filename OPT n TO output-file
```

Although this manual lists all the parameters expected by the commands, you can display the argument template of any command by simply typing the name of the command, followed by a space and a question mark (?). The command responds by displaying the argument template and waiting for you to enter the correct arguments.

If the arguments you specify do not match the template, most commands simply display the message 'Bad args' or 'Bad arguments' and stop. You must retype the command name and argument. To display on the screen help on what arguments the command expected, you can always type a question mark (?).

1.5 Restart Validation Process

When you first insert a disk for updating, AmigaDOS creates a process at low priority. This validates the entire structure on the disk. Until the restart process has completed this job, you cannot create files on the disk. It is possible, however, to read files.

When the restart process completes, AmigaDOS checks to see if you have set the system date and time. To set the date and time, you use the DATE command. If you do not specify the system date, AmigaDOS sets the system date to the date and time of the most recently created file on the inserted disk. This ensures that newer versions of files have more recent dates, even though the the actual time

and date will be incorrect.

If you ask for the date and the time before the validation is complete, AmigaDOS displays the date and time as unset. You can then either wait for the validation to complete or use DATE to enter the correct date and time. Validation should happen at once; otherwise, it should never take longer than one minute.

1.6 An Example Session

The following is an example of a simple session using AmigaDOS. The actual screen interaction with AmigaDOS is indented to distinguish it from text describing the action. Note also that whatever the computer displays is printed in a bold typeface to distinguish it from anything you might type.

```
> CD
  sys:
```

If you use the CD command without any further qualification, AmigaDOS displays the name of the current directory. (Remember that > is the usual prompt from AmigaDOS.)

```
> CD df1:tim
```

You can also use CD to change the current directory. The command sequence listed above made the directory 'tim' on disk 'df1:'. the new current directory. To gain access to files stored in this directory, you simply type the filename. You no longer need to refer to the directory structure.

```
> LIST
temp          Dir  rwed Today  08:57:16
book          Dir  rwed Today  16:39:30
doc           Dir  rwed Today  09:46:06
bench1        111  rwed Today  17:08:22
bench2        125  rwed Today  18:14:24
```

LIST requests an extended list of all the files held in the current directory (df1:tim). There are two files and three directories in this directory. The directories have the word 'Dir' in the second column; the files have their size in the second column. The letters 'r', 'w', 'e', and 'd' refer to the protection status of the particular file or directory. The letter 'r' means that you can read the file or directory, 'w' means that you write to it, 'e' means that you can execute it, and 'd' means that you can delete it. (Currently, AmigaDOS only uses the 'd' flag.) LIST uses the last two columns to indicate when you created a file or directory.

```
> CD doc
```

The name used here after CD has no colon (:) before it, and so AmigaDOS makes the search for the name from the current directory rather than from the root of a filing system. The current directory is now 'df1:tim/doc'. To look at the files stored in this directory, you would use the following command:

```
> LIST
```

to display the following:

```
plan          420 rwed Today 10:06:47
chapter1     2300 rwed Today 11:45:07
```

You can use COPY to create the file 'contents' in the directory 'df1:tim/doc', and everything you type at the terminal goes into the file until you press CTRL-\. This sends a new line and end-of-file character and terminates the file.

```
> copy * to contents
The AmigaDOS User's Manual

Chapter 1: Introduction to AmigaDOS
CTRL-\
```

You can then examine the directory contents again to see that the file does indeed exist.

```
> LIST
contents       63 rwed Today  17:01:46
plan          420 rwed Today  10:06:47
chapter1     2300 rwed Today  11:45:07
```

To see what is in the file called 'contents', you can instruct AmigaDOS to display the file by giving the following command:

```
> type contents
```

AmigaDOS then displays the contents of 'contents':

```
The AmigaDOS User's Manual

Chapter 1: Introduction to AmigaDOS
```

Chapter 2: AmigaDOS Commands

This chapter is divided into two parts: the first part describes the user commands available on the Amiga; the second describes the developer commands. The user commands fall into several categories: file utilities, CLI control, command sequence control, and system and storage management. Part 1 provides alphabetized command descriptions that give the format, template, purpose, and specification of each command as well as an example of its use. Part 2 has the same organization.

The chapter starts with a list of unfamiliar terminology. At the end of the chapter there is a quick reference card that lists all the commands by function.

Table of Contents

2.1 AmigaDOS User's Commands

Unfamiliar Terminology

In this manual you could find some terms that you have not seen before. The list below includes some common terms that are confusing if you are unfamiliar with them.

Boot startup. It comes from the expression "pulling yourself up by your bootstraps."

Default initial setting or, in other words, what happens if you do nothing. So that, in this manual, 'default' is used to mean 'in absence of something else.'

Device name part of a name that precedes the colon (:), for example, CON:, DF0:, PRT:, and so forth.

File handle an internal AmigaDOS value that represents an open file or device.

Logical device a name you can give to a directory with ASSIGN that you can then use as a device name.

Object code binary output from an assembler or compiler, and binary input to a linker.

Reboot restart.

Stream an open file or device that is associated with a file handle. For example, the input stream could be from a file and the output stream could be to the console device.

System disk a disk containing the Workbench and commands.

Volume name a name you give to a physical disk.

;

Format: [<command>];[<comment>]

Template: "command";"comment"

Purpose: To add comments to command lines.

Specification:

The CLI ignores everything after the semicolon (;).

Examples:

```
; This line is only a comment
```

ignores the part of the line containing "This line is only a comment."

```
copy <file> to prt: ; print the file
```

copies the file to the printer, but ignores the comment "print the file."

See also:

EXECUTE

> <

Format: <command> [>][<][<arg>*]

Template: "command" >"TO" <"FROM" "args"

Purpose: To direct command input and output.

Specification:

You use the symbols > and < to direct the output and input of a command. The direction of the point of the angle bracket indicates the direction of information flow. You can use these symbols to change where any command reads input or writes output. The output from a command usually goes to the current window. However, if you type a > symbol after a command and before a filename, the command writes the output to that file instead. Similarly, if you type the < symbol before a filename, the command reads from that file instead of from the keyboard.

You do not have to specify both the TO and FROM directions and files. The existence and number of "args" depends on the command you used. Redirection only happens for the command you specified. AmigaDOS reverts to the initial or 'default' input and output (that is, the keyboard and current window) afterwards.

Examples:

```
DATE > diary_dates
```

writes the output of the DATE command (that is, today's date and time) to the file 'diary_dates'.

```
my_program < my_input
```

tells my_program to acccept input from my_input instead of from the keyboard.

```
LIST > temp
SORT temp TO *
```

produces a sorted list of files and displays them on the screen.

The following sequence:

```
ECHO > 2nd.date 02-jan-78
DATE < 2nd.date ?
DELETE 2nd.date
```

creates a file called 2nd.date that contains the text "02-jan-78<linefeed>". Next it uses this file as input to the command DATE. Note that the '?' is necessary for DATE to accept input from the standard input, rather than the command line. Finally, as you no longer need the file, the DELETE command deletes 2nd.date.

ASSIGN

Format: ASSIGN [[<name>] <dir>] [LIST]

Template: ASSIGN "NAME,DIR,LIST/S"

Purpose: To assign a logical device name to a filing system directory.

Specification:

NAME is the logical device name given to the directory specified by DIR.

If you just give the NAME, AmigaDOS deletes the logical device name given (that is, it removes the assignment).

ASSIGN without any parameters or the switch LIST displays a listing of all current assignments.

When you use ASSIGN, you must ensure that there is a disk inserted in the drive. This is important because ASSIGN makes an assignment to a disk and not to a drive.

Note that the effect of ASSIGN is lost when you restart or 'reboot' your computer.

Examples:

```
ASSIGN sources: :new/work
```

Sets up the logical device name 'sources' to the directory ':new/work'. Then to gain access to files in ':new/work', you can use the logical device name 'sources', as in

```
TYPE sources:xyz
```

which displays the file ':new/work/xyz'.

```
ASSIGN LIST
```

lists the current logical device names in use.

BREAK

Format: BREAK <task> [ALL] [C] [D] [E] [F]

Template: BREAK "TASK/A,ALL/S,C/S,D/S,E/S,F/S"

Purpose: To set attention flags in the given process.

Specification:

BREAK sets the specified attention flags in the process. C sets the CTRL-C flag, D sets the CTRL-D flag, and so on. ALL sets all the flags from CTRL-C through CTRL-F. By default, AmigaDOS only sets the CTRL-C flag. The action of BREAK is identical to selecting the relevant process by moving the mouse to the window, clicking the Selection Button, and pressing the required control key combination.

Examples:

```
BREAK 7
```

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

```
BREAK 5 D
```

sets the CTRL-D attention flag of process 5.

```
BREAK 3 D E
```

sets both CTRL-D and CTRL-E.

CD

Format: CD [<dir>]

Template: CD "DIR"

Purpose: To set or change a current directory or drive.

Specification:

CD with no parameters displays the name of the current directory. In the format list above, <dir> indicates a new current directory (that is, one in which unqualified filenames are looked up). If the directory you specify is not on the current drive, then CD also changes the current drive.

To change the current directory to the directory that owns the current one (if one exists), type CD followed by a single slash (/). Thus CD/ moves the current directory one level up in the hierarchy unless the current directory is a root directory (that is, the top level in the filing system). Multiple slashes are allowed; each slash refers to an additional level above.

Examples:

```
CD df1:work
```

sets the current directory to 'work' on disk 'df1', and sets the current drive to 'df1'.

```
CD SYS:COM/BASIC
CD /
```

sets the current directory to 'SYS:COM'.

COPY

Format: COPY [[FROM] <name>] [TO <name>] [ALL] [QUIET]

Template: COPY "FROM,TO/A,ALL/S,QUIET/S"

Purpose: To copy a file or directory from one place to another.

Specification:

COPY places a copy of the file or directory in the file or directory specified as TO. The previous contents of TO, if any, are lost.

If you specify a directory name as FROM, COPY copies all the files in the FROM directory to the TO directory. If you do not specify the FROM directory, AmigaDOS uses the current directory. The TO directory must exist for COPY to work; it is not created by COPY.

If you specify ALL, COPY also copies the files in any subdirectories. In this case, it automatically creates subdirectories in the TO directory, as required. The name of the current file being copied is displayed on the screen as it happens unless you give the QUIET switch.

You can also specify the source directory as a pattern. In this case, AmigaDOS copies any files that match the pattern. See the command LIST for a full description of patterns. You may specify directory levels as well as files as patterns.

Examples:

```
COPY file1 TO :work/file2
```

copies 'file1' in the current directory to 'file2' in the directory ':work'.

```
COPY TO df1:backup
```

copies all the files in the current directory to 'df1:backup'. It does not copy any subdirectories, and DF1:backup must already exist.

```
COPY df0: to df1: ALL QUIET
```

makes a logical copy of disk 'df0' on disk 'df1' without any reflection of filenames.

```
COPY test-#? to df1:xyz
```

copies all files in the current directory that start 'test-' to the directory xyz on the disk 'df1', assuming that 'xyz' already exists. (For an explanation of patterns, such as '#?', see the command LIST in this chapter.)

```
COPY test_file to PRT:
```

copies the file 'test_file' to your printer.

```
COPY * TO CON:10/10/200/100/
```

sends everything you type at the keyboard to the console device. Press CTRL-\ to finish.

```
COPY DF0:?/#? TO DF1: ALL
```

copies every file in any 1 character subdirectory of DF0: to the root directory of DF1:.

See also:

JOIN

DATE

Format: DATE [<date>] [<time>] [TO|VER <name>]

Template: DATE "DATE,TIME,TO=VER/K"

Purpose: To display or set the system date or time.

Specification:

DATE with no parameter displays the currently set system date and time. This includes the day of the week.

DATE <date> sets the date. The form of <date> is DD-MMM-YY. If the date is already set, you can reset it by specifying a day name (this sets the date forward to that day) or by specifying 'tomorrow' or 'yesterday'.

DATE <time> sets the time. The form of <time> is HH:MM (for Hours and Minutes). You should use leading zeros when necessary. Note that, if you use a colon (:), AmigaDOS recognizes that you have specified the time rather the date. That is to say, you can set both the date and the time or either and in any order because DATE only refers to the time when you use the form HH:MM.

If you do not set the date, the restart disk validation process sets the system date to the date of the most recently created file. See Chapter 1 for details on the Restart Validation Process.

To specify the destination of the verification, you use the equivalent keywords TO and VER. The destination is the terminal unless you specify otherwise.

Note: If you type DATE before the restart validation has completed, the time is displayed as unset. To set the time, you can either use DATE or just wait until the validation process is finished.

Examples:

```
DATE
```

displays the current date.

```
DATE 06-Sep-82
```

sets the date to 6th September 1982, the time to 00:00.

```
DATE tomorrow
```

resets the date to one day ahead.

```
DATE TO fred
```

sends the current date to the file "fred".

```
DATE 10:50
```

sets the current time to ten 'til eleven.

```
DATE 23:00
```

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

```
DATE 01-JAN-02
```

sets the date to January 1st, 2002. (The earliest date you can set is 01-JAN-78.)

DELETE

Format: DELETE <name> [<name>*][ALL][Q|QUIET]

Template: DELETE ",,,,,,,,,ALL/S,Q=QUIET/S"

Purpose: To delete up to ten files or directories.

Specification:

DELETE attempts to delete each file you specify. If it cannot delete a file, the screen displays a message, and AmigaDOS attempts to delete the next file in the list. You may not delete a directory if it contains any files.

You can also use a pattern to specify the filename. See the description of the command LIST for full details of patterns. The pattern may specify directory levels as well as filenames. In this case, all files that match the pattern are deleted.

If you specify ALL, DELETE also deletes the files in any subdirectories.

Unless you specify the switch QUIET (or use the alternative, Q), the name of the file being deleted appears on the screen as it happens.

Examples:

```
DELETE old-file
```

deletes the file 'old-file'.

```
DELETE work/prog1 work/prog2 work
```

deletes the files 'prog1' and 'prog2' in the directory 'work', and then deletes the directory 'work'.

```
DELETE t#?/#?(1/2)
```

deletes all the files that end in '1' or '2' in directories that start with 't'. (For an explanation of patterns, such as '#?', see the command LIST later in this chapter.)

```
DELETE DF1:#? ALL
```

deletes all the files on DF1:.

See also:

DIR (I-DEL option)

DIR

Format: DIR [<name>] [OPT A|I|AI]

Template: DIR "DIR,OPT/K"

Purpose: To provide a display of the files in a directory in sorted order. DIR can also include the files in subdirectories, and you can use DIR in interactive mode.

Specification:

DIR alone shows the files in the current directory. DIR followed by a directory provides the files in that directory. The form of the display is first any subdirectories, followed by a sorted list of the files in two columns.

To pass options to DIR, use the OPT keyword. Use the A option to include any subdirectories below the specified one in the list. Each sublist of files is indented.

To list only the directory names use the D option.

The I option specifies that DIR is to run in interactive mode. In this case, the files and directories are displayed with a question mark following each name. Press RETURN to display the next name in the list. To quit the program, type Q. To go back to the previous directory level or to stop (if at the level of the initial directory), type B.

If the name displayed is that of a directory, type E to enter that directory and display the files and subdirectories. Use E and B to select different levels. Typing the command DEL can be used to delete a directory, but this only works if the directory is empty.

If the name is that of a file, typing DEL deletes the file, or typing T Types (that is, displays) the file at the screen. In the last case, press CTRL-C to stop it 'typing' and return to interactive mode.

To find the possible responses to an interactive request, type ?.

Examples:

```
DIR
```

provides a list of files in current directory.

```
DIR df0: OPT a
```

lists the entire directory structure of the disk 'df0'.

```
DIR df0:com OPT i
```

provides an interactive listing of the directory 'df0:com'.

DISKCOPY

Format: DISKCOPY [FROM] <disk> TO <disk> [NAME <name>]

Template: DISKCOPY "FROM/A,TO/A/K,NAME/K"

Purpose: To copy the contents of one 3-1/2 inch floppy disk to another.

Specification:

DISKCOPY makes a copy of the entire contents of the disk you specified as FROM, overwriting the previous contents of the entire disk you specified as TO. DISKCOPY also formats a new disk as it copies. You normally use the command to produce backup floppy disks.

Once you have given the command, AmigaDOS prompts you to insert the correct disks. At this point, you insert the correct source and destination disks.

You can use the command to copy any 3-1/2 inch AmigaDOS disk to another, but the source and destination disks must be identical in size and structure. To copy information between different sized disks, you use COPY.

You can also use the command to copy a floppy disk using a single floppy drive. If you specify the source and destination as the same device, then the program reads in as much of the source disk into memory as possible. It then prompts you to place the destination disk in the drive and then copies the information from memory onto the destination disk. This sequence is repeated as many times as required.

If you do not specify a new name for your disk, DISKCOPY creates a new disk with the same name as the old one. However, AmigaDOS can tell the difference between two disks with the same name because every disk is associated with the date and time of its creation. DISKCOPY gives the new disk the current system date as its creation date and time.

Note: To copy part of a disk, you can use COPY to RAM:.

Examples:

```
DISKCOPY FROM df0: TO df1:
```

makes a backup copy of the disk 'df0' onto disk 'df1'.

```
DISKCOPY FROM df0: TO df0:
```

makes a backup copy of the disk in drive 'df0' using only a single drive.

See also:

COPY

ECHO

Format: ECHO <string>

Template: ECHO ""

Purpose: To display the argument given.

Specification:

ECHO writes the single argument to the current output stream (which can be a file or a device). This is normally only useful within a command sequence or as part of a RUN command. If you give the argument incorrectly, an error is displayed.

Examples:

```
RUN COPY :work/prog to df1:work ALL QUIET +
ECHO "Copy finished"
```

creates a new CLI to copy the specified directory as a background process. When it has finished, the screen displays

Copy finished

ED

Format: ED [FROM] <name> [SIZE <n>]

Template: ED "FROM/A,SIZE"

Purpose: To edit text files.

Specification:

ED is a screen editor. You can use ED as an alternative to the line editor EDIT. The file you specify as FROM is read into memory, then ED accepts your editing instructions. If FROM does not exist, AmigaDOS creates a new file.

Because the file is read into memory, there is a limit to the size of file you can edit with ED. Unless you specify otherwise, workspace size is 40,000 bytes. This workspace size is usually sufficient for most files. However, to alter the workspace, you specify a suitable value after the SIZE keyword.

There is a full specification of ED in Chapter 3.

Examples:

```
ED work/prog
```

edits the file 'work/prog', assuming it exists; otherwise, ED creates the file.

```
ED huge-file SIZE 50000
```

edits a very large file 'huge-file', using a workspace of 50,000 bytes.

EDIT

Format: EDIT [FROM] <name> [[TO] <name>][WITH <name>][VER <name>][OPT <option>]

Template: EDIT "FROM/A,TO,WITH/K,VER/K,OPT/K"

Purpose: To edit text files.

Specification:

EDIT is a line editor (that is, it edits a sequential file line by line). If you specify TO, EDIT copies from file FROM to file TO. Once you have completed the editing, the file TO contains the edited result, and the file FROM is unchanged. If you do not specify TO, then EDIT writes the edited text to a temporary file. If you give the EDIT commands Q or W, then EDIT renames this temporary file FROM, having first saved the old version of FROM in the file ':t/edit-backup'. If you give the EDIT command STOP, then EDIT makes no change to the file FROM.

EDIT reads commands from the current input stream, or from a WITH file if it specified.

EDIT sends editor messages and verification output to the file you specify with VER. If you omit VER, the terminal is used instead.

OPT specifies options: Pn sets the maximum number of previous lines to n; Wn sets the maximum line width. The initial setting is P40W120.

Note: You cannot use the < and > symbols to redirect input and output when you call EDIT.

See Chapter 4 for a full specification of EDIT.

Examples:

```
EDIT work/prog
```

edits the file 'work/prog'. When editing is complete, EDIT saves the old version of 'work/prog' in ':t/edit-backup'.

```
EDIT work/prog TO work/newprog
```

edits the file 'work/prog', placing the edited result in the file 'work/newprog'.

```
EDIT work/prog WITH edits/0 VER nil:
```

edits the file 'work/prog' with the edit commands stored in the file 'edits/0'. Verification output from EDIT is sent to the dummy device 'nil:'.

ENDCLI

Format: ENDCLI

Template: ENDCLI

Purpose: To end an interactive CLI process.

Specification:

AmigaDOS only allows ENDCLI as an interactive command. ENDCLI removes the CLI currently selected by the mouse.

You shouldn't use ENDCLI except on a CLI created by the NEWCLI command. If the initial CLI (process 1) is ended, and no other has been set up by the NEWCLI command, then the effect is to terminate the AmigaDOS session.

Note that there are no arguments to the ENDCLI command, and no check for invalid arguments.

Note: Do not experiment with ENDCLI before you've used NEWCLI. Using ENDCLI on the initial CLI always pulls the rug out from under you by terminating that CLI. If you started the CLI from the Workbench, then there is no problem as you are returned to the Workbench. If you started AmigaDOS with just the CLI running, then ending the last CLI gives you no way of creating a new one.

Examples:

The following sequence:

```
NEWCLI
LIST
ENDCLI
```

opens a new window, lists the directory, and closes the window again.

EXECUTE

Format: EXECUTE <commandfile> [<arg>*]

Template: EXECUTE "command-file","args"

Purpose: To execute a file of commands with argument substitution.

Specification:

You normally use EXECUTE to save typing. The command-file contains commands executed by the Command Line Interface. AmigaDOS executes these commands one at a time, just as though you had typed them at the keyboard.

You can also use EXECUTE to perform parameter (that is, value) substitution, where you can give certain names as parameters. Before the command file is executed, AmigaDOS checks the parameter names with those you've given after the EXECUTE command. If any match, AmigaDOS uses the values you specified instead of the parameter name. Parameters may have values specified that AmigaDOS uses if you do not explicitly set the parameter. If you have not specified a parameter, and if there is no default, then the value of the parameter is empty and nothing is substituted for it.

To use parameter substitution, you give directives to the EXECUTE command. To indicate these, you start a line with a special character, which is initially a period or 'dot' (.). The directives are as follows:

.KEY	)	Argument template, used to specify
.K	)	the format of the arguments
.DOT	ch	Change dot character (initially '.') to ch
.BRA	ch	Change bra character (initially '<') to ch
.KET	ch	Change ket character (initially '>') to ch
.DOL	)	
.DOLLAR)	ch	Change default-char (initially '$') to ch
.DEF	keyword value	Give default to parameter
.<space>		Comment line
.<newline>		Blank comment line

Before execution, AmigaDOS scans the contents of the file for any items enclosed by BRA and KET characters ('<' and '>'). Such items may consist of a keyword or a keyword and a default value for AmigaDOS to use if you have left the keyword unset. (To separate the keyword and the default, if there is one, you type a dollar sign '$'). Thus, AmigaDOS replaces <ANIMAL> with the value you associated with the keyword ANIMAL, while it replaces <ANIMAL$WOMBAT> with the value of ANIMAL if it has one, and otherwise it defaults to WOMBAT.

AmigaDOS provides a number of commands that are only useful in command sequence files. These include IF, SKIP, LAB, and QUIT. These can be nested in a command file.

Note that you can also nest EXECUTE files. That is, you can have a command file that contains EXECUTE commands.

To stop the execution of a command file, you press CTRL-D.

Examples:

Assume the file 'list' contains the following:

.k file/a
```
run copy <file> to prt:+
echo "Printing of <file> done"
```

Then the following command

```
EXECUTE list test/prg
```

acts as though you had typed the following commands at the keyboard.

```
RUN copy test/prg to prt:+
ECHO "Printing of test/prg done"
```

Another example, "display", uses more of the features described above:

```
.key file/a
IF EXISTS <file>
TYPE <file> OPT n
. If the file given is on the current directory, type it
. with line numbers.
ELSE
ECHO "<file> is not on this directory"
ENDIF
.

RUN EXECUTE display work/prg2
```

should display the file work/prg2 with line numbers on the terminal if it exists on the current directory. If the file is not there, the screen displays the following message:

```
work/prg2 is not on this directory.
```

See also:

;, IF, SKIP, FAILAT, LAB, ECHO, RUN, QUIT

FAILAT

Format: FAILAT <n>

Template: FAILAT "RCLIM"

Purpose: To instruct a command sequence to fail if a program returns an error code greater than or equal to this number.

Specification:

Commands indicate that they have failed in some way by setting a return code. A non-zero return code indicates that the command has found an error of some sort. A return code greater than or equal to a certain limit (the fail limit) terminates a sequence of non-interactive commands (that is, the commands that you specify after RUN or in an EXECUTE file). The return code indicates how serious the error was, and is normally 5, 10 or 20.

You may use the FAILAT command to alter this fail level from its initial value of 10. If you increase the level, 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 should be a positive number. The fail level is reset to the initial value of 10 on exit from the command sequence.

You must use FAILAT before commands such as IF to test to see if a command has failed; otherwise, the command sequence terminates before executing the IF command.

If you omit the argument, the current value of the fail level is displayed.

Examples:

```
FAILAT 25
```

The command sequence only terminates before the end if a command stops with a return code greater than or equal to (>=) 25.

See also:

IF, EXECUTE, RUN, QUIT

FAULT

Format: FAULT [<n>*]

Template: FAULT ",,,,,,,,"

Purpose: To display the messages corresponding to the fault codes you supply.

Specification:

AmigaDOS looks up the numbers and displays the corresponding messages. Up to ten messages may be displayed.

Examples:

```
FAULT 182
```

displays the message for fault 182.

```
FAULT 294 296 199
```

displays the messages for faults 294, 296, and 199.

FILENOTE

Format: FILENOTE [FILE] <file> COMMENT <string>

Template: FILENOTE "FILE/A,COMMENT/K"

Purpose: To attach a comment or a note to a file.

Specification:

FILENOTE assigns a comment to a specified file.

The keyword COMMENT introduces an optional comment of up to 80 characters. A comment may be more than one word (that is, contain spaces between characters). In this case, you must enclose the comment within double quotes (").

A comment is associated with a particular file. When you examine the file with the command LIST, the comment appears on the line below:

```
prog              30 rwed  Today  11:07:33
 : version 3.2 - 23-mar-85
```

When you create a new file, it does not normally have a comment. If you overwrite an existing file that has a comment, then the comment is retained even though the contents of the file has changed. The command COPY copies a file. If a file with a comment is copied, the new file does not have the comment from the original attached to it although the destination file may have a comment which is retained.

Examples:

```
FILENOTE prog2  COMMENT "Ver 3.3 26-mar-85"
```

attaches the comment "Ver 3.3 26-mar-85" to program 2.

See also:

LIST

FORMAT

Format: FORMAT DRIVE <drivename> NAME <string>

Template: FORMAT "DRIVE/A/K,NAME/A/K"

Purpose: To format and initialize a new 3 1/2 inch floppy disk.

Specification:

The program formats a new floppy disk in the method required for AmigaDOS. Once the disk is formatted, it is initialized and assigned the name you specify. Notice that you must give both the DRIVE and NAME keywords. The only valid options that you can give after the DRIVE keyword are DF0:, DF1:, DF2:, or DF3:. You can type any string after NAME, but if you use spaces, you must enclose the whole string in double quotes (").

WARNING: FORMAT formats and initializes a disk as an empty disk. If you use a disk that is not empty, you'll lose the previous contents of the disk.

The name assigned should be unique. It may be one to thirty characters in length and composed of one or more words separated by spaces. If the name is more than one word, you should enclose it in double quotes.

Note: It is not necessary to format a disk if you are about to DISKCOPY to it.

Examples:

```
FORMAT DRIVE df0: NAME "Work disk"
```

formats and initializes the disk in drive 'df0' with the name "Work disk".

See also:

DISKCOPY, INSTALL, RELABEL

IF

Format: IF [NOT] [WARN] [ERROR] [FAIL] [<str> EQ <str>] [EXISTS <name>]

Template: IF "NOT/S,WARN/S,ERROR/S,FAIL/S,,EQ/K,EXISTS/K"

Purpose: To allow conditionals within command sequences.

Specification:

You can only use this command in an EXECUTE command file. If one or more of the specified conditions is satisfied, IF carries out all the following commands until it finds a corresponding ENDIF or ELSE command; otherwise, if the conditions are not satisfied, it carries out whatever follows a corresponding ELSE command. (ENDIF and ELSE are only useful in command sequences containing IF.) ENDIF terminates an IF command; ELSE provides an alternative if the IF conditions fail. Note that the conditions and commands in an IF and ELSE command can span more than one line before their corresponding ENDIF.

The following table shows some of the ways you can use the IF, ELSE, and ENDIF commands:

```
IF <condition>          IF <condition>          IF <condition>
<command>               <command>               <command>
ENDIF                   ELSE                    IF <condition>
                        <command>               <command>
                        ENDIF                   ENDIF
                                                ENDIF
```

Note that ELSE is optional and that nested IFs jump to the nearest ENDIF.

ERROR is only available if you set FAILAT to greater than 10. Similarly, FAIL is only available if you set FAILAT to greater than 20.

Keyword	Function
NOT	reverses the result.
WARN	satisfied if previous return code >= 5.
ERROR	" " " " " >= 10.
FAIL	" " " " " >= 20.
<a> EQ <b>	satisfied if the text of a and b is identical (disregarding case).
EXISTS <file>	satisfied if the file exists

You can use IF EQ to detect an unset parameter in a command file by using the form

```
IF <a> EQ ""
```

Examples:

```
IF EXISTS work/prog
TYPE work/prog
ELSE
ECHO "file not found"
ENDIF
```

If the file 'work/prog' exists, then AmigaDOS displays it. Otherwise, AmigaDOS displays the message "file not found" and executes the next command in the command sequence.

```
IF ERROR
SKIP errlab
ENDIF
```

If the previous command stopped with a return code >= 10, then AmigaDOS skips the command sequence until you define a label 'errlab' with the LAB command.

```
IF ERROR
IF EXISTS fred
ECHO "The file 'fred' exists, but an error occurred anyway."
ENDIF
ENDIF
```

See also:

FAILAT, SKIP, LAB, EXECUTE, QUIT

INFO

Format: INFO

Template: INFO

Purpose: To give information about the filing system.

Specification:

The command displays a line of information about each disk unit. This includes the maximum size of the disk, the current used and free space, the number of soft disk errors that have occurred, and the status of the disk.

Examples:

```
INFO

Unit  Size  Used  Free  Full  Errs  Status       Name
DF1:  880K     2  1756    0%  0     Read/Write   Test-6
DF0:  880K  1081   677   61%  0     Read/Write   AmigaDOS CLI V27.5 4-Jul-85
RAM:    1K     1     0  100%  0     Read/Write

Volumes available:
Test-6 [Mounted]
AmigaDOS CLI V27.5 4-Jul-85 [Mounted]
```

INSTALL

Format:

INSTALL [DRIVE] <drive>

Template:

INSTALL "DRIVE/A"

Purpose: To make a formatted disk bootable.

Specification:

The purpose of the INSTALL command is to make a disk bootable (that is, you can use INSTALL to make a disk that starts up your Amiga). To do this, you simply type the name of the drive where you have inserted the disk that you want to become the boot (startup) disk. There are four possible drive names: DF0:, DF1:, DF2:, and DF3:.

Examples:

```
INSTALL df0:
```

makes the disk in drive 'df0:' a bootable disk.

JOIN

Format: JOIN <name> <name> [<name>*] AS <name>

Template: JOIN ",,,,,,,,,,,,,,,AS/A/K"

Purpose: To concatenate up to 15 files to form a new file.

Specification:

AmigaDOS copies the specified files in the order you give into the new file. Note that the new file cannot have the same name as any of the input files.

Examples:

```
JOIN part1 part2 AS textfile
```

joins the two files together, placing the result in 'textfile'. The two original files remain unchanged, while 'textfile' contains a copy of 'part1' and a copy of 'part2'.

LAB

Format: LAB <string>

Template: LAB <text>

Purpose: To implement labels in command sequence files.

Specification:

The command ignores any parameters you give. Use LAB to define a label 'text' that is looked for by the command SKIP.

Examples:

```
LAB errlab
```

defines the label 'errlab' to which SKIP may jump.

See also:
SKIP, IF, EXECUTE

LIST

Format: LIST [DIR] <dir> [P|PAT <pat>] [KEYS] [DATES] [NODATES] [TO <name>] [S <str>] [SINCE <date>] [UPTO <date>] [QUICK]

Template: LIST "DIR,P=PAT/K,KEYS/S,DATES/S,NODATES/S,TO/K,S/K, SINCE/K,UPTO/K,QUICK/S"

Purpose: To examine and list specified information about a directory or file.

Specification:

If you do not specify a name (the parameter DIR), LIST displays the contents of the current directory. The first parameter LIST accepts is DIR. You have three options. DIR may be a filename, in which case LIST displays the file information for that one file. Secondly DIR may be a directory name. In this case LIST displays file information for files (and other directories) within the specified directory. Lastly, if you omit the DIR parameter, LIST displays information about files and directories within the current directory (for further details on the current directory, see the CD command).

Note: LIST, unlike DIR, does NOT sort the directory before displaying it.

If no other options are specified, LIST displays

```
file_name      size  protection date time
:comment
```

These fields are defined as follows:

file_name: Name of file or directory.

size: The size of the file in bytes. If there is nothing in the file, this field will state "empty". For directories this entry states "dir".

protection: This specifies the access available for this file. rwed indicates Read, Write, Execute, and Delete.

date and time: The file creation date and time.

comment: This is the comment placed on the file using the FILENOTE command. Note that it is preceded with a colon (:).

Options available:

TO This specifies the file (or device) to output the file listing to. If omitted, the output goes to the current CLI window.

KEYS displays the block number of each file header or directory.

DATES	displays dates in the form DD-MMM-YY (the default unless you use QUICK).
NODATES	does not display date and time information.
SINCE <date>	displays only files last updated on or after <date>. <date> can be in the form DD-MMM-YY or a day name in the last week (for example, MONDAY) or TODAY or YESTERDAY.
UPTO <date>	displays only files last updated on or before <date>.
P <pat>	searches for files whose names match <pat>.
S <str>	searches for filenames containing substring <str>.
QUICK	just displays the names of files and directories (like the DIR command).

You can specify the range of filenames displayed in two ways. The simplest way is to use the S keyword, which restricts the listing to those files containing the specified substring. To specify a more complicated search expression, use the P or PAT keyword. This is followed by a pattern that matches as described below.

A pattern consists of a number of special characters with special meanings, and any other characters that match themselves.

The special characters are: '()?%#|

In order to remove the special effect of these characters, preface them with '. Thus '? matches ? and '' matches '.

?	matches any single character.
%	matches the null string.
#<p>	matches zero or more occurrences of the pattern <p>.
<p1><p2>	matches a sequence of pattern <p1> followed by <p2>.
<p1>|<p2>	matches if either pattern <p1> or pattern <p2> match.
()	groups patterns together.

Thus:

A#BC	matches AC ABC ABBC, and so forth.
A#(B|C)D	matches AD ABD ABCD, and so forth.
A?B	matches AAB ABB ACB, and so forth.
A#?B	matches AB AXXB AZXQB, and so forth.
'?#?'#	matches ?# ?AB# ??##, and so forth
A(B|%)#C	matches A ABC ACCC, and so forth.
#(AB)	matches AB ABAB ABABAB, and so forth.

Examples:

```
LIST
```

displays information about all the files and directories contained in the current directory. For example,

```
File_1
File 2
File.3
:comment
File004
```

notice that File.3 has a comment.

```
LIST work S new
```

displays information about files in the directory 'work' whose names contain the text 'new'.

```
LIST work P new#?(x|y)
```

examines the directory 'work', and displays information about all files that start with the letters 'new' and that end with either 'x' or 'y'.

```
LIST QUIET TO outfile
```

sends just the names, one on each line, to the file 'outfile'. You can then edit the file and insert the command TYPE at the beginning of each line. Then type

```
EXECUTE outfile
```

to display the files.

See also:

DATE, DIR, FILENOTE, PROTECT

MAKEDIR

Format: MAKEDIR <dir>

Template: MAKEDIR "/A"

Purpose: To make a new directory.

Specification:

MAKEDIR creates a directory with the name you specify. The command only creates one directory at a time, so any directories on the path must already exist. The command fails if the directory or a file of the same name already exists in the directory above it in the hierarchy.

Examples:

```
MAKEDIR tests
```

creates a directory 'tests' in the current directory.

```
MAKEDIR df1:xyz
```

creates a directory 'xyz' in the root directory of disk 'df1'.

```
MAKEDIR df1:xyz/abc
```

creates a directory 'abc' in the parent directory 'xyz' on disk 'df1'. However, 'xyz' must exist for this command to work.

See also:

DELETE

NEWCLI

Format: NEWCLI [<window>]

Template: NEWCLI "WINDOW"

Purpose: To create a window associated with a new interactive CLI process.

Specification:

AmigaDOS creates a new CLI window. The new window becomes the currently selected process. The new window has the same set directory and prompt string as the one where NEWCLI is executed. Each CLI window is independent, allowing separate input, output, and program execution.

To connect the keyboard to your new CLI, move the mouse to point the cursor at the new window, and press the left mouse button (that is, the Selection Button). You can point at any position on the window when selecting a new CLI.

When you give NEWCLI with no argument, AmigaDOS creates a window of standard size and position. To change the size of the window, move the mouse to point the cursor at the bottom right corner (sizing Gadget), and press the Selection Button. You can then change the window size. To change the position of the window, move the mouse to the Drag Bar, press the left mouse button and move the mouse to where you want the window.

To customize a CLI window, you can give an exact position and size or even a new title on the title bar. The 'window' syntax to do this is as follows:

CON:x/y/width/height/title

where 'CON:' denotes a console window, 'x' and 'y' are the coordinates describing the window's position, 'width' and 'height' are the size of the window, and 'title' is the string you want on the title bar. You need not specify a title string as it is optional, but you must give the final slash (/). All dimensions are in screen pixels.

Examples:

```
NEWCLI
```

creates a new CLI process and makes it the current CLI.

```
NEWCLI CON:10/30/300/100/myCLI
```

creates a new CLI at the position 10,30, of size 300x100 pixels, with the title 'myCLI'.

```
NEWCLI "CON:20/15/300/100/my own CLI"
```

Double quotes allow the title to have spaces. For further information on the console device, CON:, see Section 1.3.6 Understanding Device Names.

Note: Unlike a background process created with the RUN command, a NEWCLI process hangs around after you have created it.

See also:

ENDCLI, RUN

PROMPT

Format: PROMPT <prompt>

Template: PROMPT "PROMPT"

Purpose: To change the prompt in the current CLI.

Specification:

If you do not give a parameter, then AmigaDOS resets the prompt to the standard string ("> "). Otherwise, the prompt is set to the string you supply. AmigaDOS also accepts one special character combination (%N). This is demonstrated in the example below.

Examples:

```
PROMPT
```

resets the current prompt to "> ".

```
PROMPT "%N> "
```

resets the current prompt to "n> ", where n is the current process number. AmigaDOS interprets the special character combination %N as the process number.

PROTECT

Format: PROTECT [FILE] <filename> [FLAGS <status>]

Template: PROTECT "FILE,FLAGS/K"

Purpose: To set a file's protection status.

Specification:

PROTECT takes a file and sets its protection status.

The keyword STATUS takes four options: read (r), write (w), delete (d), and execute (e). To specify these options you type an r, w, d, or e after the name of the file. If you omit an option, PROTECT assumes that you do not require it. For instance, if you give all the options except d, PROTECT ensures that you cannot delete the file. Read, write, and delete can refer to any kind of file. You only use execute in reference to files containing executable programs, not **source code**. When you specify execute (e), the program can be executed.

Examples:

```
PROTECT prog1 STATUS r
```

sets the protection status of program 1 as read only.

```
PROTECT prog2 rwd
```

sets the protection of program 2 as read/write/delete.

See also:

LIST

QUIT

Format: QUIT [<returncode>]

Template: QUIT "RC"

Purpose: To exit from a command sequence with a given error code.

Specification:

QUIT reads through the command file and then stops with a return code. The default return code is zero.

Examples:

```
QUIT
```

exits the current command sequence.

```
FAILAT 30
IF ERROR
QUIT 20
ENDIF
```

If the last command was in error, this terminates the command sequence with return code 20.

For more on command sequences, see the specification for the EXECUTE command earlier in this chapter.

See also:

EXECUTE, IF, LAB, SKIP

RELABEL

Format: RELABEL [DRIVE] <drive> [NAME] <name>

Template: RELABEL "DRIVE/A,NAME/A"

Purpose: To change the volume name of a disk.

Specification:

RELABEL changes the volume name of a disk to the <name> you specify. Volume names are set initially when you format a disk.

Examples:

```
RELABEL df1: "My other disk"
```

See also:

FORMAT

RENAME

Format: RENAME [FROM] <name> [TO|AS] <name>

Template: RENAME "FROM/A,TO=AS/A"

Purpose: To rename a file or directory.

Specification:

RENAME renames the FROM file with the specified TO name. FROM and TO must be filenames on the same disk. The FROM name may refer to a file or to a directory. If the filename refers to a directory, RENAME leaves the contents of the directory unchanged (that is, the directories and files within that directory keep the same contents and names).

Only the name of the directory is changed when you use RENAME. If you rename a directory, or if you use RENAME to give a file another directory name (for example, rename :bill/letter as :mary/letter), AmigaDOS changes the position of the directory, or file, in the filing system hierarchy. Using RENAME is like changing the title of a file and then moving it to another section or drawer in the filing cabinet. Some other systems describe the action as 'moving' a file or directory.

Note: If you already have a file with exactly the same name as the TO file, RENAME won't work. This should stop you from overwriting your files by accident.

Examples:

```
RENAME work/prog1 AS :arthur/example
```

renames the file 'work/prog1' as the file 'arthur/example'. The root directory must contain 'arthur' for this command to work.

RUN

Format: RUN <command>

Template: RUN command+
command .....

Purpose: To execute commands as background processes.

Specification:

RUN creates a non-interactive Command Line Interface (CLI) process and gives it the rest of the command line as input. The background CLI executes the commands and then deletes itself.

The new CLI has the same set directories and command stack size as the CLI where you called RUN.

To separate commands, type a plus sign (+) and press RETURN. RUN interprets the next line after a + (RETURN) as a continuation of the same command line. Thus, you can make up a single command line of several physical lines that each end with a plus sign.

RUN displays the process number of the newly created process.

Examples:

```
RUN COPY :t/0 PRT:+
DELETE :t/0+
ECHO "Printing finished"
```

prints the file ':t/0' by copying it to the line printer device, deletes it, then displays the given message.

```
RUN EXECUTE comseq
```

executes in the background all the commands in the file 'comseq'.

SEARCH

Format: SEARCH [FROM] <name>|<pat> [SEARCH] <string> [ALL]

Template: SEARCH "FROM,SEARCH/A,ALL/S"

Purpose: To look for a text string you specify in all the files in a directory.

Specification:

SEARCH looks through all the files in the specified directory, and any files in subdirectories if you specify ALL. SEARCH displays any line that contains the text you specified as SEARCH. It also displays the name of the file currently being searched.

You can also replace the directory FROM with a pattern. (See the command LIST for a full description of patterns.) If you use a pattern, SEARCH only looks through files that match the specified pattern. The name may also contain directories specified as a pattern.

AmigaDOS looks for either upper or lower case of the search string. Note that you must place quotation marks around any text containing a space.

As usual, to abandon the command, press CTRL-C, the attention flag. To abandon the search of the current file and continue on to the next file, if any, press CTRL-D.

Examples:

```
SEARCH SEARCH vflag
```

searches through the files in the current directory looking for the text 'vflag'.

```
SEARCH df0: "Happy day" ALL
```

looks for files containing the text 'Happy day' on the entire disk 'df0:'.

```
SEARCH test-#? vflag
```

looks for the text 'vflag' in all files in the current directory starting with 'test-'.

SKIP

Format: SKIP <label>

Template: SKIP "LABEL"

Purpose: To perform a jump in a command sequence.

Specification:

You use SKIP in conjunction with LAB. (See LAB for details.) SKIP reads through the command file looking for a label you defined with LAB, without executing any commands.

You can use SKIP either with or without a label; without one, it finds the next unnamed LAB command. With one, it attempts to find a LAB defining a label, as specified. LAB must be the first item on a line of the file. If SKIP does not find the label you specified, the sequence terminates and AmigaDOS displays the following message:

label "<label>" not found by Skip

SKIP only jumps forwards in the command sequence.

Examples:

```
SKIP
```

skips to the next LAB command without a name following it.

```
IF ERROR

SKIP errlab

ENDIF
```

If the last command stopped with a return code >= 20, this searches for the label 'errlab' later in the command file.

```
FAILAT 100
ASSEM text
IF ERROR
SKIP ERROR
ENDIF
LINK
SKIP DONE
LAB ERROR
ECHO "Error doing Assem"
LAB DONE
ECHO "Next command please"
```

See also:

EXECUTE, LAB, IF, FAILAT, QUIT

SORT

Format: SORT [FROM] <name> [[TO] <name>] [COLSTART <n>]

Template: SORT "FROM/A,TO/A,COLSTART/K"

Purpose: To sort simple files.

Specification:

This command is a very simple sort package. You can use SORT to sort files although it isn't fast for large files, and it cannot sort files that don't fit into memory.

You specify the source as FROM, and the sorted result goes to the file TO. SORT assumes that FROM is a normal text file where each line is separated with a carriage return. Each line in the file is sorted into increasing alphabetic order without distinguishing between upper and lower cases.

To alter this in a very limited way, use the COLSTART keyword to specify the first column where the comparison is to take place. SORT then compares the characters on the line from the specified starting position to the end; if the lines still match after this, then the remaining columns from the first to just before the column specified as COLSTART are included in the comparison.

Note: The initial stack size (that is, 4000 bytes) is only suitable for small files ofless than 200 lines or so. If you want to sort larger files, you must use the STACK command to increase the stack size; how much you should increase the size is part skill and part guesswork.

WARNING: The Amiga will crash if STACK is too small. If you are not sure, it is better to overestimate the amount you need.

Examples:

```
SORT text TO sorted-text
```

sorts each line of information in 'text' alphabetically and places the result in 'sorted-text'.

```
SORT index TO sorted-index COLSTART 4
```

sorts the file 'index', where each record contains the page number in the first three columns and the index entry on the rest of the line, and puts the output in 'sorted-index' sorted by the index entry, and matching index entries sorted by page number.

See also:

><, STACK

STACK

Format: STACK [<n>]

Template: STACK "SIZE"

Purpose: To display or set the stack size for commands.

Specification:

When you run a program, it uses a certain amount of stack space. In most cases, the initial stack size is sufficient, but you can alter it using the STACK command. To do this, you type STACK followed by the new stack value. You specify the value of the stack size in bytes. STACK alone displays the currently set stack size.

The only command that you would normally need to alter the stack size for is the SORT command. Recursive commands such as DIR need an increased stack if you use them on a directory structure more than about six levels deep.

WARNING: The only indication that you have run out of stack is that the Amiga crashes! If you are not sure, it is better to overestimate the amount you need.

Examples:

STACK

displays the current stack size.

STACK 8000

sets the stack to 8000 bytes.

See also:

RUN, SORT

STATUS

Format: STATUS [<process>] [FULL] [TCB] [SEGS] [CLI|ALL]

Template: STATUS "PROCESS,FULL/S,TCB/S,SEGS/S,CLI=ALL/S"

Purpose: To display information about the currently existing CLI processes.

Specification:

STATUS alone lists the numbers of the CLI processes and the program running in each.

PROCESS specifies a process number and only gives information about that process. Otherwise, information is displayed about all processes.

FULL = SEGS + TCB + CLI

SEGS displays the names of the sections on the segment list of each process.

TCB displays information about the priority, stacksize, and global vector size of each process.

For further details on stack and global vector size, see the *AmigaDOS Technical Reference Manual*.

CLI identifies Command Line Interface processes and displays the section name(s) of the currently loaded command (if any).

Examples:

```
STATUS
```

displays brief information about all processes.

```
STATUS 4 FULL
```

displays full information about process 4.

TYPE

Format: TYPE [FROM] <name> [[TO] <name>] [OPT N|H]

Template: TYPE "FROM/A,TO,OPT/K"

Purpose: To type a text file or to type a file out as hexadecimal numbers.

Specification:

TO indicates the output file that you specify; if you omit this, output is to the current output stream, which means, in most cases, that the output goes to the current window.

Tabs that you have given in the file are expanded. However, tabs are not treated as special by TYPE; the console driver processes them. To interrupt output, press CTRL-C. To suspend output, press the space bar or type any other character. To resume output, press RETURN or CTRL-X.

OPT specifies an option to TYPE. The first option to TYPE is 'n', which includes line numbers in the output.

The second option you can give TYPE is h. Use the h option to write out each word of the FROM file as a hex number.

Examples:

```
TYPE work/prog
```

displays the file 'work/prog'.

```
TYPE work/prog OPT n
```

displays the file 'work/prog' with line numbers.

```
TYPE obj/prog OPT h
```

displays the code stored in 'obj/prog' in hexadecimal.

WAIT

Format: WAIT <n> [SEC|SECS] [MIN|MINS] [UNTIL <time>]

Template: WAIT ",SEC=SECS/S,MIN=MINS/S,UNTIL/K"

Purpose: To wait for the specified time.

Specification:

You can use WAIT in command sequences or after RUN to wait for a certain period, or to wait until a certain time of day. Unless you specify otherwise, the waiting time is one second.

The parameter should be a number, specifying the number of seconds (or minutes, if MINS is given) to wait.

Use the keyword UNTIL to wait until a specific time of day, given in the format HH:MM.

Examples:

```
WAIT
```

waits 1 second.

```
WAIT 10 MINS
```

waits 10 minutes.

```
WAIT UNTIL 21:15
```

waits until quarter past nine at night.

WHY

Format: WHY

Template: WHY

Purpose: To explain why the previous command failed.

Specification:

Usually when a command fails the screen displays a brief message that something went wrong. This typically includes the name of the file (if that was the problem), but does not go into any more detail. For example, the command

```
COPY fred TO *
```

might fail and display the message

```
Can't open fred
```

This could happen for a number of reasons - for example, 'fred' might already be a directory, or there might not be enough space on the disk to open the file, or it might be a read-only disk. COPY makes no distinction between these cases, because usually the user knows what is wrong. However, immediately after you come across a command that has failed, you can type WHY and press RETURN to display a much fuller message, describing in detail what went wrong.

Examples:

```
WHY
```

gives information about why the last command failed.

See also:

FAULT

2.2 AmigaDOS Developer's Commands

ALINK

Format: ALINK [[FROM|ROOT] <filename> [,<filename>* | + <filename*]]
[TO <name>] [WITH <name>][LIBRARY|LIB <name>]
[MAP <map>] [XREF <name>] [WIDTH <n>]

Template: ALINK "FROM=ROOT,TO/K,WITH/K,VER/K,LIBRARY=LIB/K,MAP/K,
XREF/K,WIDTH/K"

Purpose: To link together sections of code into an executable file.

Specification:

ALINK instructs AmigaDOS to link files together. It also handles automatic library references and builds overlay files. The output from ALINK is a file loaded by the loader and run under the overlay supervisor, if required.

For details and a full specification of the ALINK command, see Chapter 4 of the *AmigaDOS Developer's Manual*.

Examples:

```
ALINK a+b+c TO output
```

links the files 'a', 'b' and 'c', producing an output file 'output'.

ASSEM

Format: ASSEM [PROG|FROM] <prog> [-O <code>] [-V <ver>] [-L <listing>] [-E] [-C|OPT <opt>] [-I <dirlist>]

Template: ASSEM "PROG=FROM/A,-O/K,-V/K,-L/K,-H/K,-E/K,-C=OPT/K,-I/K"

Purpose: To assemble a program in MC68000 assembly language.

Specification:

ASSEM assembles programs in MC68000 assembly language. See Chapter 3 of the *AmigaDOS Developer's Manual* for details.

PROG	is the source file.
-O	is the object file (that is, binary output from the assembler)
-V	is the file for messages (unless you specify -V, messages go to the terminal)
-L	is the listing file.
-C	specifies options to the assembler.
-H	is a header file which can be read as if inserted at the front of the source (like INCLUDE in the source itself).
-I	sets up a list of directories to be searched for included files.
-E	is the file that receives the 'equates' directive (EQU) assignments from your source. You use -E to generate a header file containing these directives.

The options you can specify with OPT or -C are as follows:

S	produce a symbol table dump as part of the object file.
X	produce a cross-reference file.
W<size>	set workspace to <size>.

Examples:

```
ASSEM prog.asm TO prog.obj
```

assembles the source program in 'prog.asm', placing the result in the file 'prog.obj'. It writes any error messages to the terminal, but does not produce an assembly listing.

```
ASSEM prog.asm TO prog.obj -h slib -l prog-list
```

assembles the same program to the same output, but includes the file 'slib' in the assembly, and places an assembly listing in the file 'prog-list'.

```
ASSEM foo.asm -o foo.obj opt w8000
```

assembles a very small program.

DOWNLOAD

Template: DOWNLOAD "FROM/A,TO/A"

Purpose: To download programs to the Amiga.

Specification:

The command DOWNLOAD downloads programs written on another computer (for example, a Sun) to the Amiga.

To use DOWNLOAD, you must have a BillBoard. Then, to download your linked load file from the Sun to the Amiga, you type on the Sun

binload -p &

(this only needs to be done once), then type on the Amiga

download <sun filename> <amiga filename>

(Before you boot your Sun, you must make sure that both the BillBoard and Amiga are already on and powered up, otherwise they won't be recognized by the Sun.) The <sun filename> by convention should end with .ld. Once you've done this, to run the program, you type the <amiga filename>.

Note that the command 'binload' is not an AmigaDOS command. You use binload on a Sun to load files in binary for downloading to your Amiga.

Note that DOWNLOAD always accesses files on the Sun relative to the directory in which binload was started. If you cannot remember the directory in which binload was started, you must specify the full name must be specified. To stop binload on the Sun, you can do a 'ps' and then a 'kill' on its PID. Note that the soft reset of the computer tells binload to write a message to its standard output (the default being the window where it started). If DOWNLOAD hangs, press CTRL-C to kill it.

Chapter 1 of the *AmigaDOS Developer's Manual* describes in detail how to download programs from an IBM PC to Amiga, from the Sun to the Amiga, and even gives some hints on how to download from unsupported computers.

Examples:

```
binload -p &

download test.ld test.
```

or

```
download /usr/fred/DOS/test.ld test
```

then type the following:

```
test
```

These commands download the specified Sun filenames to the Amiga filenames.

READ

Template: READ "TO/A,SERIAL/S"

Purpose: READ reads data from the parallel port or serial line and stores it in a file.

Specification:

The command READ listens to the parallel port and expects a stream of hexadecimal characters. If you give the SERIAL switch, READ listens, instead, to the serial line. Each hex pair is stored as a byte in memory. READ recognizes Q as the hex stream terminator. READ also recognizes the ASCII digits 0-9 and the capital letters A through F. READ ignores spaces, new lines, and tabs. You must send an ASCII hex digit for every nibble, and you must have an even number of nibbles. When the stream is complete, READ writes the bytes from memory to the disk file you specified.

Note: You can use this command to transfer binary or text files.

WARNING 1: Be careful when READing to the same file twice. READ overwrites the original contents the second time.

WARNING 2: You may lose characters if you use high baud rates with the serial connection.

Examples:

```
READ TO df0:new
```

READs to the file 'df0:new' from the parallel port.

```
READ new SERIAL
```

READs to the file 'new' from the serial line.

AmigaDOS Commands Quick Reference Card

User's Commands

File Utilities

;	comment character
< >	direct command input and output respectively.
COPY	copies one file to another or copies all the files from one directory to another.
DELETE	deletes up to 10 files or directories.
DIR	shows filenames in a directory.
ED	enters a screen editor for text files.
EDIT	enters a line by line editor.
FILENOTE	attaches a note with a maximum of 80 characters to a specified file.
JOIN	concatenates up to 15 files to form a new file.
LIST	examines and displays detailed information about a file or directory.
MAKEDIR	creates a directory with a specified name.
PROTECT	sets a file's protection status.
RENAME	renames a file or directory.
SEARCH	looks for a specified text string in all the files of a directory.
SORT	sorts simple files.
TYPE	types a file to the screen that you can optionally specify as text or hex.

CLI Control

BREAK	sets attention flags in a given process.
CD	sets a current directory and/or drive.
ENDCLI	ends an interactive CLI process.
NEWCLI	creates a new interactive CLI process.

PROMPT	changes the prompt in the current CLI.
RUN	executes commands as background processes.
STACK	displays or sets the stack size for commands.
STATUS	displays information about the CLI processes currently in existence.
WHY	explains why a previous command failed.

Command Sequence Control

ECHO	displays the message specified in a command argument.
EXECUTE	executes a file of commands.
FAILAT	fails a command sequence if a program returns an error code greater than or equal to this number.
IF	tests specified actions within a command sequence.
LAB	defines a label (see SKIP).
QUIT	exits from a command sequence with a given error code.
SKIP	jumps forward to LAB in a command sequence (see LAB).
WAIT	waits for, or until, a specified time.

System and Storage Management

ASSIGN	assigns a logical device name to a filing system directory.
DATE	displays or sets the system date and time.
DISKCOPY	copies the contents of one entire floppy disk to another.
FAULT	displays messages corresponding to supplied fault or error codes.
FORMAT	formats and initializes a new 3 1/2 inch floppy disk.
INFO	gives information about the filing system.
INSTALL	makes a formatted disk bootable.
RELABEL	changes the volume name of a disk.

Developer's Commands

Development System

ALINK links sections of code into a file for execution (see JOIN).

ASSEM assembles M68000 language.

DOWNLOAD downloads programs to the Amiga.

READ reads information from the parallel port or serial line and stores it in a file.

Chapter 3: ED - The Screen Editor

This chapter describes how to use the screen editor ED. You can use this program to alter or create text files.

Table of Contents

3.1 Introducing ED

You can use the editor ED to create a new file or to alter an existing one. You display text on the screen, and you can scroll it vertically or horizontally, as required.

ED accepts the following template:

```
ED "FROM/A,SIZE/K"
```

For example, to call ED, you type

```
ED fred
```

ED makes an attempt to open the file you have specified as 'fred' (that is, the FROM file), and if this succeeds, then ED reads the file into memory and displays the first few lines on the screen. Otherwise, ED provides a blank screen, ready for the addition of new information. To alter the text buffer that ED uses to hold the file, you specify a suitable value after the SIZE keyword, for example,

```
ED fred SIZE 45000
```

The initial size is based on the size of the file you edit, with a minimum of 40,000 bytes.

Note: You cannot edit every kind of file with ED. For example, ED does not accept source files containing binary code. To edit files such as these, you should use the editor EDIT.

WARNING: ED always appends a linefeed even if the file does not end with one.

When ED is running, the bottom line of the screen is a message area and command line. Error messages appear here and remain until you give another ED command.

ED commands fall into two categories:

- immediate commands
- extended commands.

You use immediate commands in **immediate mode**; you use extended commands in **extended mode**. ED is already in immediate mode when you start editing. To enter extended mode, you press the ESC key. Then, after ED has executed the command line, it returns automatically to immediate mode.

In immediate mode, ED executes commands right away. You specify an immediate command with a single key or control key combination. To indicate a control key combination, you press and hold down the CTRL key while you type the given letter, so that CTRL-M, for example, means hold down CTRL while you type M.

In extended mode, anything you type appears on the command line. ED does not execute commands until you finish the command line. You may type a number of extended commands on a single command line. You may also group any commands together and even get ED to repeat them automatically. Most immediate commands have a matching extended version.

ED attempts to keep the screen up to date. However, if you enter a further command while it is attempting to redraw the display, ED executes the command at once and updates the display when there is time. The current line is always displayed first and is always up to date.

3.2 Immediate Commands

This section describes the type of commands that ED executes immediately. Immedi ate commands deal with the following:

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

3.2.1 Cursor Control

To move the cursor one position in any direction, you press the appropriate curs or control key. If the cursor is on the right hand edge of the screen, ED scrolls the text to the left to make the rest of the text visible. ED scrolls vertically a 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 hand edge of the text.

CTRL-], that is, CTRL and the square closing bracket ']' takes the cursor to the right hand edge of the current line unless the cursor is already there. When the cursor is already at the right hand edge, CTRL-] moves it back to the left hand edge of the line. The text is scrolled horizontally, if required. In a similar fashion, CTRL-E places the cursor at the start of the first line on the screen unless the cursor is already there. If the cursor is already there, CTRL-E places it at the end of the last line on the screen.

CTRL-T takes the cursor to the start of the next word. CTRL-R takes the cursor to the space following the previous word. In these two cases, the text is scrolled vertically or horizontally, as required.

The TAB key moves the cursor to the next tab position, which is a multiple of the tab setting (initially 3). It does NOT insert TAB characters in the file.

3.2.2 Inserting Text

Any letter that you type in immediate mode appears at the **current cursor position** unless the line is too long (there is a maximum of 255 characters in a line). If you try to make a line longer than the maximum limit, ED refuses to add another character and displays the following message:

```
Line too long
```

However, on shorter lines, ED moves any characters to the right of the cursor to make room for the new text. If the line exceeds the size of the screen, the right hand end of the line disappears from view. Then, ED redisplays the end of the line by scrolling the text horizontally. If you move the cursor beyond the end of the line, for example, with the TAB or cursor control keys, ED inserts spaces between the end of the line and any new character you insert.

To split the current line at the cursor and generate a new line, press RETURN. If the cursor is at the end of a line, ED creates a new blank line after the current one. Alternatively, you press CTRL-A to generate a blank line after the current one, with no split of the current line taking place. In either case,

the cursor appears on the new line at the position indicated by the left margin (initially, column one).

To ensure that ED gives a carriage return automatically at a certain position on the screen, you can set up a right margin. Once you have done this, whenever you type a line that exceeds that margin, ED ends the line before the last word and moves the word and the cursor down onto a new line. (Note that if you have a line with no spaces, ED won't know where to break the 'word' and the automatic margin cannot work properly.) In detail, if you type a character and the cursor is at the end of the line and at the right margin position, then ED automatically generates a new line. Unless the character you typed was a space, ED moves down the half completed word at the end of the line to the newly generated line. However, if you insert some text when the cursor is NOT at the end of a line (that is, with text already to the right of the cursor), then setting a right margin does not work. Initially, the right margin is set up at column 79. You can turn off, or 'disable,' the right margin with the EX command. (For further details on setting margins, see Section 3.3.1 Program Control).

If you type some text in the wrong case (for example, in lower case instead of upper case), you can correct it with CTRL-F. To do this, you move the cursor to point at the letter you want to change and then press CTRL-F. If the letter is in lower case, CTRL-F flips the letter into upper case. On the other hand, if the letter is in upper case, CTRL-F flips it into lower case. However, if the cursor points at something that is not a letter (for example, a space or symbol), CTRL-F does nothing to it.

CTRL-F not only flips letter cases but it also moves the cursor one place to the right (and it moves the cursor even if there is no case to flip). So that, after you have changed the case of a letter with CTRL-F, the cursor moves right to point at the next character. If the next character is a letter, you can press CTRL-F again to change its case; you can then repeat the command until you have changed all the letters on the line. (Note that if you continue to press CTRL-F after the the last letter on the line, the cursor keeps moving right even though there is nothing left to change.) For example, if you had the line

```
The Walrus and the Carpenter were walking hand in hand
```

and you kept CTRL-F pressed down, the line would become

```
tHE wALRUS AND THE cARPENTER WERE WALKING HAND IN HAND
```

On the other hand, the following line:

```
IF <file> <= x
```

becomes

```
if <FILE> <= X
```

where the letters change case and the symbols remain the same.

3.2.3 Deleting Text

The BACKSPACE key deletes the character to the left of the cursor and moves the cursor one position left unless it is at the beginning of a line. ED scrolls the text, if required. The DEL key deletes the character at the current cursor position without moving the cursor. As with any deletion, characters remaining on the line shift back, and text that was invisible beyond the right hand edge of the screen becomes visible.

The action of CTRL-O depends on the character at the cursor. If this character is a space, then CTRL-O deletes all spaces up to the next non-space character on the line. Otherwise, it deletes characters from the cursor, and moves text left, until a space occurs.

CTRL-Y deletes all characters from the cursor to the end of the line.

CTRL-B deletes the entire current line. You may use extended commands to delete blocks of text.

3.2.4 Scrolling

Besides vertically scrolling one line line at a time by moving the cursor to the edge of the screen, you can vertically scroll the text 12 lines at a time with the control keys CTRL-U and CTRL-D.

CTRL-D moves the cursor to previous lines, while scrolling the text down; CTRL-U scrolls the text up and moves the cursor to lines further on in the file.

CTRL-V refreshes the entire screen, which is useful if another program besides the editor alters the screen. However, in typical use, messages from other processes appear in the window behind the editor window.

3.2.5 Repeating Commands

The editor remembers any extended command line you type. To execute this set of extended commands again at any time, press CTRL-G. In this way, you can set up a search command as an extended command. If the first occurrence of a string is not the one you need, press CTRL-G to repeat the search. You can set up and execute complex sets of editing commands many times.

Note: When you give an extended command as a command group with a repetition count, ED repeats the commands in the group that number of times each time you press CTRL-G. See the section "Repeating Commands" for more details on extended commands.

3.3 Extended Commands

This section describes the commands available to you in extended mode. These commands cover

- program control
- block control
- movement
- searching text
- exchanging text
- altering text
- inserting text

To enter extended command mode, press the ESC key. Subsequent input then appears on the command line at the bottom of the screen. You can correct mistakes with BACKSPACE in the normal way. To terminate the command line, press either ESC or RETURN. If you press ESC, the editor remains in extended mode after executing the command line. On the other hand, if you press RETURN, it reverts to immediate mode. To leave the command line empty, just press RETURN after pressing ESC to go back to immediate mode. In this case, ED returns to immediate command mode.

Extended commands consist of one or two letters, with upper and lower case considered the same. You can give multiple commands on the same command line by separating them with a semicolon. Commands are sometimes followed by an argument, such as a number or a string. A string is a sequence of letters introduced and terminated by a delimiter, which is any character except letters, numbers, space, semicolon, or brackets. Thus, valid strings might be

```
/happy/   !23 feet!   :Hello!: "1/2"
```

Most immediate commands have a corresponding extended version. See the Table of Extended Commands at the end of this chapter for a complete list.

3.3.1 Program Control

This section provides a specification of the program control commands X (eXit), Q (Quit), SA (SAve), U (Undo), SH (SHow), ST (Set Tab), SL and SR (Set Left and Set Right), and EX (EXtend).

To instruct the editor to exit, you use the command X. After you have given the exit command, ED writes out the text it is holding in memory to the output, or destination file and then terminates. If you look at this file, you can see that all the changes you made are there.

ED also writes a temporary backup to :T/ED-BACKUP. This backup file remains until you exit from ED again, at which time, ED overwri tes the file with a new backup.

To get out of the editor without keeping any changes, you use the Q command. When you use Q, ED terminates immediately without writing to the buffer and discards any changes you have made. Because of this, if you have altered the contents of the file, ED asks you to confirm that you really want to quit.

A further command lets you to take a 'snapshot' copy of the file without coming out of ED. This is the SA command. SA saves the text to a named file or, in the absence of a named file , to the current file. For example,

```
SA !:doc/savedtext!
```

or

```
SA
```

SA is particularly useful in geographical areas subject to power failure or surge.

Hint: SA followed by Q is equivalent to the X command.

If you make any alterations between the SA and the Q commands, the following message appears:

```
Edits will be lost - type Y to confirm:
```

If you have made no alterations, ED quits immediately with the contents of your source file unchanged. SA is also useful b ecause it allows you to specify a filename other than the current one. It is therefore possible to make copies at different stages and place them in different files or directories.

To undo the last change, you use the U command. The editor makes a copy of the line the cursor is on, and then it modifies this copy whenever you add or delete characters. ED puts the changed copy back into the file when you move the cursor off the current line (either by cursor control, or by deleting or inserting a line). ED also replaces the copy when it performs any scrolling either vertically or horizontally. The U command discards the changed copy and uses the old version of the current line instead.

WARNING: ED does not undo a line deletion. Once you have moved from the current line, the U command cannot fix the mess you have got yourself into.

To show the current state of the editor, you use the SH command. The screen displays information such as the value of tab stops, current margins, block marks, and the name of the file being edited.

Tabs are initially set at every three columns. To change the current setting of tabs, you use the ST command followed by a number n, which sets tabs at every n columns.

To set the left margin and right margin, you use the SL and SR commands, again followed by a number indicating the column position. The left margin should not be set beyond the width of the screen.

To extend margins, you use the EX command. Once you have given EX, ED takes no account of the right margin on the current line. Once you move the cursor from the current line, ED turns the margins on again.

3.3.2 Block Control

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

You can identify a block of text with the BS (Block Start) and BE (Block End) commands. To do this, move the cursor to anywhere on the first line you that you want in the block and give the BS command. Then, move the cursor to the last line that you want in the block, using the cursor control commands or a search command, and give the BE command to mark the end of the block.

Note: Once you have defined a block with BS and BE, if you make ANY change to the text, the start and end of the block become undefined once more. The only exception to this is if you use IB (Insert Block).

To identify one line as the current block, move to the line you want, press ESC, and type

```
BS;BE
```

The current line then becomes the current block.

Note: You cannot start or finish a block in the middle of a line. To do this, you must first split the line by pressing RETURN.

Once you have identified a block, you can move a copy of it into another part of the file with the IB (Insert Block) command. When you give the IB command, ED inserts a copy of the block immediately after the current line. You can insert more than one copy of the block, as it remains defined until you change the text, or delete the block.

To delete a block, you use the DB (Delete Block) command. DB deletes the block of text you defined with the BS and BE commands. However, when you have deleted the block, the block start and end values become undefined. This means that you cannot delete a block and then insert a copy of it (DB followed by IB); however, you can insert a copy of the block and then delete the block (IB followed by DB).

You can also use block marks to remember a place in a file. The SB (Show Block) command resets the screen window on the file so that the first line in the block is at the top of the screen.

To write a block to another file, you use the WB command (Write Block). This command takes a string that represents a file name. For example,

```
WB !:doc/example!
```

writes the contents of the block to the file 'example' in the directory ':doc'. (Remember: if you use the filename-divider slash (/) to separate directories and files, you should not use slash as a delimiter). ED then creates a file with the name that you specified, possibly destroying a previous file with that name and finally writes the buffer to it.

To insert a file into the current file, you use the IF command (Insert File). ED reads into memory the file with the name you gave as the argument string to IF, at the point immediately following the current line. For example,

```
IF !:doc/example!
```

inserts the file :doc/example into the current file beginning immediately after the current line.

3.3.3 Moving the Current Cursor Position

The command T moves the cursor to the top of the file, so that the first line in the file is the first line on the screen. The B command moves the cursor to the bottom of the file, so that the last line in the file is the bottom line on the screen.

The commands N and P move the cursor to the start of the next line and previous line, respectively. The commands CL and CR move the cursor one place to the left or one place to the right, while CE places the cursor at the end of the current line, and CS places it at the start.

The command M moves the cursor to a specific line. To move, you type M followed by the line number of the line you want as the new current line. For example,

```
M 503
```

moves the cursor to the five hundred and third line in the file. The M command is a quick way of reaching a known position in your file. You can, for instance, move to the correct line in your file by giving a repeat count to the N command, but it is much slower.

3.3.4 Searching and Exchanging

Alternatively you can move the screen window to a particular context with the command F (Find) followed by a string that represents the text to be located. The search starts at one place beyond the current cursor position and continues forwards through the file. If the string is found, the cursor appears at the start of the located string.

To search backwards through the text, you use the command BF (Backwards Find) in the same way as F. BF finds the last occurrence of the string before the current cursor position. (That is, BF looks for the string to the left of the cursor and then through all the lines back to the beginning of the file.) To find the earliest occurrence, you use T (Top-of-file) followed by F. To find the last occurrence, you use B (Bottom-of-file) followed by BF.

The E (Exchange) command takes two strings separated with delimiter characters and exchanges the first string for the last. So, for example,

```
E /wombat/zebra/
```

would change the next occurrence of the text 'wombat' to 'zebra'. The editor starts searching for the first string at the current cursor position and continues through the file. After the exchange is completed, the cursor moves to the end of the exchanged text.

You can specify empty strings by typing two delimiters with nothing between them. If the first, or 'search', string is empty, the editor inserts the second string at the current cursor position. If the second string is empty, the next occurrence of the search string is exchanged for nothing (that is, the search string is deleted).

Note: ED ignores margin settings while you are exchanging text.

The EQ command (Exchange and Query) is a variant on the E command. When you use EQ, ED asks you whether you want the exchange to take place. This is useful when you want the exhange to take place in some circumstances, but not in others. For example, after typing

```
EQ /wombat/zebra/
```

the following message

Exchange?

appears on the command line. If you respond with an N, then the cursor moves past the search string; otherwise, if you type Y, the change takes place as normal. You usually only give EQ in repeated groups.

The search and exchange commands usually make a distinction between upper and lower case while making the search. To tell all subsequent searches not to make any distinction between upper and lower case, you use the UC command. Once you have given UC, the search string "wombat" matches "Wombat", "WOMBAT", "WoMbAt" and so on. To have ED distinguish between upper and lower case again, you use LC.

3.3.5 Altering Text

You cannot use the E command to insert a new line into the text. You use the I and A commands instead. Follow the I command (Insert before) with a string that you want to make into a new line. ED inserts this new line 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. You use the A command (insert After) in the same way except that ED inserts the new line after the current line. That is,

```
A /Insert this AFTER the current line/
```

inserts the string "Insert this AFTER the current line" as a new line After the line containing the cursor.

To split the current line at the cursor position, you use the S command. S in extended mode is just like pressing RETURN in immediate mode (see Section 3.2.2 for further details on splitting lines).

The J command joins the next line onto the end of the current one.

The D command deletes the current line in the same way as CTRL-B in immediate mode. The DC command deletes the character above the cursor in the same way as DEL.

3.3.6 Repeating Commands

To repeat any command a certain number of times, precede it with the desired number. For example,

```
4 E /slithy/brillig/
```

changes the next four occurrences of 'slithy' to 'brillig'. ED verifies the screen after each command. You use the RP (Repeat) command to repeat a command until ED returns an error, such as reaching the end of the file. For example,

```
T; RP E /slithy/brillig/
```

changes all occurrences of 'slithy' to 'brillig'. Notice that you need the T command to ensure that all occurrences of 'slithy' are changed, otherwise only those after the current position are changed.

To execute command groups repeatedly, you can group the commands together in parentheses. You can also nest command groups within command groups. For example,

```
RP ( F /bandersnatch/; 3 A/ / )
```

inserts three blank lines (copies of the null string) after every line containing 'bandersnatch'. Notice that this command line only works from the cursor to the end of the file. To apply the command to every line in the file, you should first move to the top of the file.

Note that some commands are possible, but silly. For example,

```
RP SR 60
```

sets the right margin to 60 *ad infinitum*. However, to interrupt any sequence of extended commands, and particularly repeated ones, you type any character while the commands are taking place. If an error occurs, ED abandons the command sequence.

Quick Reference Card

Special Key Mappings

Command	Action
BACKSPACE	Delete character to left of cursor
DEL	Delete character as cursor
ESC	Enter extended command mode
RETURN	Split line at cursor and create a new line
TAB	Move cursor right to next tab position (does NOT insert a TAB character)
	Move cursor up
	Move cursor down
	Move cursor left
	Move cursor right

Immediate Commands

Command	Action
CTRL-A	Insert line
CTRL-B	Delete line
CTRL-D	Scroll text down
CTRL-E	Move to top or bottom of screen
CTRL-F	Flip case
CTRL-G	Repeat last extended command line
CTRL-H	Delete character left of cursor (BACKSPACE)
CTRL-I	Move cursor right to next tab position
CTRL-M	Return
CTRL-O	Delete word or spaces
CTRL-R	Cursor to end of previous word
CTRL-T	Cursor to start of next word
CTRL-U	Scroll text up
CTRL-V	Verify screen
CTRL-Y	Delete to end of line
CTRL-[	Escape (enter extended mode)
CTRL-]	Cursor to end or start of line

Extended Commands

This is a full list of extended commands including those that are merely extended versions of immediate commands. In the list, /s/ indicates a string, /s/t/ indicates two exchange strings, and n indicates a number.

Command	Action
A /s/	Insert line after current
B	Move to bottom of file
BE	Block end at cursor
BF /s/	Backwards find
BS	Block start at cursor
CE	Move cursor to end of line
CL	Move cursor one position left
CR	Move cursor one position right
CS	Move cursor to start of line
D	Delete current line
DB	Delete block
DC	Delete character at cursor
E /s/t/	Exchange s into t
EQ /s/t/	Exchange but query first
EX	Extend right margin
F /s/	Find string s
I /s/	Insert line before current
IB	Insert copy of block
IF /s/	Insert file s
J	Join current line with next
LC	Distinguish between upper and lower case in searches
M n	Move to line number n
N	Move to start of next line
P	Move to start of previous line
Q	Quit without saving text
RP	Repeat until error
S	Split line at cursor
SA	Save text to file
SB	Show block on screen
SH	Show information
SL n	Set left margin
SR n	Set right margin
ST n	Set tab distance
T	Move to top of file
U	Undo changes on current line
UC	Equate U/C and l/c in searches
WB /s/	Write block to file s
X	Exit, writing text into memory

Chapter 4: EDIT - The Line Editor

This chapter describes in detail how to use the line editor EDIT. The first part introduces the reader to the editor. The second part gives a complete specification of EDIT. There is a quick reference card containing all the EDIT commands at the end of the chapter.

Table of Contents

(continuation of the Table of Contents)

4.1 Introducing EDIT

EDIT is a text editor that processes **sequential files** line by line under the control of **editing commands**. EDIT moves through the input, or **source file**, passing each line (after any possible alterations) to a sequential output file, the **destination file**. An EDIT run, therefore, makes a copy of the source file that contains any changes that you requested with the editing commands.

Although EDIT usually processes the source file in a forward sequential manner, it has the capability to move backwards a limited number of lines. This is possible because EDIT doesn't write the lines that have been passed to the destination file immediately, but holds them instead in an **output queue**. The size of this queue depends on the amount of **memory** available. If you want to hold more information in memory, you can select the EDIT option, OPT, described in the next section, to increase the amount.

You can make more than one pass through the text.

The EDIT commands let you

a) change parts of the source,
b) output parts of the source to other destinations, and
c) insert material from other sources.

4.1.1 Calling EDIT

This section describes the format of the **arguments** you can give every time you call the EDIT command. EDIT expects the following arguments:

FROM/A,TO,WITH/K,VER/K,OPT/K

The **command template** described in Chapter 1 is a method of defining the syntax for each command. AmigaDOS accepts command arguments according to the format described in the command template. For example, some arguments are optional, some must appear with a keyword, and others do no need keywords because they only appear in a specific position. Arguments with a following /A (like FROM) must appear, but you do not have to type the keyword. Arguments with just a following /K (such as WITH, VER, and OPT) are optional, but you must type the keyword to specify them. Arguments without a following / (TO, for example) are optional. AmigaDOS recognizes arguments without a following slash (/) by their position alone. If you forget the syntax for EDIT, type

```
EDIT ?
```

and AmigaDOS displays the full template on the screen. (For more details on usin g commands, see Chapters 1 and 2 of this manual.)

Using another method of description, the command syntax for EDIT is as follows:

[FROM] <file> [[TO] <file>] [WITH <file>] [VER <file>] [OPT Pn|Wn|PnWn]

The argument FROM represents the source file that you want to edit. The argument must appear, but the keyword itself is optional (that is, AmigaDOS accepts the FROM file by its position). It does not require you to type the keyword FROM as well.

The TO file represents the destination file. This is the file where EDIT sends the output including the editing changes. If you omit the TO argument, EDIT uses a temporary file that it renames as the FROM file when editing is complete. If you give the EDIT command STOP, this renaming does not take place, and the original FROM file is untouched.

The WITH keyword represents the file containing the editing commands. If you omit the WITH argument, EDIT reads from the terminal.

The VER keyword represents the file where EDIT sends error messages and line verifications. If you omit the VER argument, EDIT uses the terminal.

You can use the OPT keyword to specify options to EDIT. Valid options are P<n>, which sets the number of previous lines available to the integer <n>, and W<n>, which sets the maximum line length handled to <n> characters. Unless you specify otherwise, AmigaDOS sets the options P40W120.

You can use OPT to increase, or decrease, the size of available memory. EDIT uses P*W (that is, the number of previous lines multiplied by the line width) to determine the available memory. To change the memory size, adjust the P and W numbers. P50 allocates more memory than usual; P30 allocates less memory than usual.

Here are some examples of how you can call EDIT:

```
EDIT programl TO programl_new WITH edit_commands

EDIT programl OPT P50W240

EDIT programl VER ver_file
```

Note: Unlike ED, you cannot use EDIT to create a new file. If you attempt to create a new file, AmigaDOS returns an error because it cannot find the new file in the current directory.

4.1.2 Using EDIT Commands

This section introduces some of the basic EDIT commands omitting many of the advanced features. A complete description of the command syntax and of all commands appears in the Section 4.2, A Complete Specification of EDIT.

4.1.2.1 The Current Line

As EDIT reads lines from the source and writes them to the destination, the line that it has "in its hand" at any time is called the current line. EDIT makes all the textual changes to the current line. EDIT always inserts new lines before the current line. When you first enter EDIT, the current line is the first line of the source.

4.1.2.2 Line Numbers

EDIT assigns each line in the source a unique line number. This line number is not part of the information stored in the file, but EDIT computes it by counting the lines as they are read. When you're using EDIT, you can refer to a specific line by using its line number. A line that has been read retains its original line number all the time it is in main memory, even when you delete lines before or after it, or insert some extra lines. The line numbers remain unchanged until you rewind the file, or until you renumber the lines with the = command. EDIT assigns the line numbers each time you enter the file. The line numbers, therefore, may not be the same when you re-enter.

4.1.2.3 Selecting a Current Line

To select a current line in EDIT, you can use one of three methods:

a) counting lines,
b) specifying the context, *or*
c) specifying the line number.

These three methods are described below.

By Line Counting

The N and P commands allow you to move to the next or previous lines. If you give a number before the N or P command, you can move that number of lines forward or backward. To move forward to the next line, type

```
N
```

For any EDIT command, you can type either upper or lower case letters.

To move four lines forward, type

```
4N
```

to make the fourth line from the current line your new current line.

To move back to a line above the current line, type

```
P
```

The P command also takes a number. For example, type

```
4P
```

This makes the fourth line above the current line your new current line. It is only possible to go back to previous lines that EDIT has not yet written to the output. EDIT usually lets you go back 40 lines. To be able to move back more than this, you specify more previous lines with the P option when you enter EDIT (see Section 4.1.1 earlier in this chapter for further details on the P option).

Moving to a Specific Line Number

The M command allows you to select a new current line by specifying its line number. You type the M command and the desired line number. For example, the command M45 tells EDIT to Move to line 45. If you are beyond line 45, this command moves back to it provided it is still in main memory.

You can combine the specific line number and line counting commands. For example,

```
M12; 3N
```

To separate consecutive commands on the same line, type ; (a semicolon).

By Context

You use the F command (Find) to select a current line by context. For example,

```
F/Jabberwocky/
```

means to find the line containing 'Jabberwocky'. The search starts at the current line and moves forward through the source until the required line is found. If EDIT reaches the end of the source without finding a matching line, it displays the following message:

```
SOURCE EXHAUSTED
```

It is also possible to search backwards by using the BF command (Backwards Find). For example,

```
BF/gyre and gimble/
```

BF also starts with the current line, but EDIT moves backwards until it finds the desired line. If EDIT reaches the head of the output queue without finding a matching line, it displays the following message:

```
NO MORE PREVIOUS LINES
```

Notice that in the examples above, the desired text (Jabberwocky and gyre and gimble) is enclosed in matching single slashes (/). This desired text is called a **character string**. The characters you use to indicate the beginning and end of the character string are called **delimiter characters**. In the examples above, / was used as the delimiter. A number of special characters such as : . , and * are available for use as delimiters; naturally, the string itself must not contain the delimiter character. EDIT ignores the spaces between the command name and the first delimiter, but considers spaces within the string as significant, since it matches the context exactly. For example,

```
F /tum tum tree/
```

does not find 'tum-tum tree' or 'tum tum tree'.

If you use an F command with no argument, EDIT repeats the previous search. For example,

```
F/jubjub bird/; N; F
```

finds the second occurrence of a line containing 'jubjub bird'. The N command between the two F commands is necessary because an F command always starts by searching the current line. If you omitted N, the second F would find the same line as the first.

4.1.2.4 Qualifiers

The basic form of the F command described above finds a line that contains the given string anywhere in its length. To restrict the search to the beginning or the end of lines, you can place one of the letters B or E in front of the string. In this case, you must type one or more spaces after F. For example,

```
F B/slithy toves/
```

means Find the line Beginning with 'slithy toves', while

```
F E/bandersnatch/
```

means Find the line Ending with 'bandersnatch'. As well as putting further conditions on the context required, the use of B or E speeds up the search, as EDIT only needs to consider part of each line.

B and E as used above are examples of **qualifiers**, and the whole argument is called a **qualified string**. A number of other qualifiers are also available. For example,

```
F P/a-sitting on a gate/
```

means Find the next line containing Precisely the text 'a-sitting on a gate'. The required line must contain no other characters, either before or after the given string. That is to say, when you give this command, EDIT finds the next line containing:

```
a-sitting on a gate
```

However, EDIT does not find the line:

```
a-sitting on a gate.
```

To find an empty line (Precisely nothing), you can use an empty string with the P qualifier, for example,

```
F P//
```

You can give more than one qualifier in any order.

4.1.2.5 Making Changes to the Current Line

This section describes how to use the E, A, and B commands to alter the text on your current line.

Exchanging strings

The E command Exchanges one string of characters in the line for another, for example:

```
E/Wonderland/Looking Glass/
```

removes the string 'Wonderland' from the current line, and replaces it with 'Looking Glass'. Note that you use a single central delimiter to separate the two strings. To delete parts of the line (exchange text for nothing), you can use a null second string, as follows:

```
E/monstrous crow//
```

To add new material to the line, you can use the A or B commands. The A command inserts its second string After the first occurrence of the first string on the current line. Similarly, the B command inserts its second string Before the first occurrence of the first string on the current line. For example, if the current line contained

```
If seven maids with seven mops
```

then the following command sequence:

```
A/seven/ty/; B L/seven/sixty-/
```

would turn it into

```
If seventy maids with sixty-seven mops
```

If you had omitted the L qualifier from the B command above, the result would be

```
If sixty-seventy maids with seven mops
```

because the search for a string usually proceeds from left to right, and EDIT uses the first occurrence that it finds. You use the qualifier L to specify that the search should proceed Leftwards. The L qualifier forces the command that it qualifies to act on the Last occurrence of its first argument.

If the first string in an A, B or E command is empty, EDIT inserts the second string at the beginning or the end of the line. To further qualify the position of the second string, you use or omit the L or the E qualifiers.

If you give EDIT an A, B, or E command on a line that does not match the qualified string given as the first argument, the following message appears either on the screen or in a verification file that you specified when you entered EDIT.

```
NO MATCH
```

See the section "Calling EDIT" for details on the verification file.

4.1.2.6 Deleting Whole Lines

This section describes how to remove lines of text from your file. To delete a range of lines, you can specify their line numbers in a D command. To use the D command, type D and the line number. If you type a space and a second number after D, EDIT removes all the lines from the first line number to the last. For example,

```
D97 104
```

deletes lines 97 to 104 inclusive, leaving line 105 as the new current line. To delete the current line, type D without a qualifying number. For example,

```
F/plum cake/; D
```

deletes the line containing 'plum cake', and the line following it becomes the new current line. You can combine a qualified search with a delete command, as follows:

```
F B/The/; 4D
```

This command sequence deletes four lines, the first of which is the line beginning with 'The'.

You can also type a period (.) or an asterisk (*) instead of line numbers. To refer to the current line, type a period. To refer to the end-of-file, type an asterisk. For example,

```
D. *
```

deletes the rest of the source including the current line.

4.1.2.7 Inserting New Lines

This section describes how to insert text into your file with EDIT. To insert one or more lines of new material BEFORE a given line, you use the I command. You can give the I command alone or with a line number, a period (.), or an asterisk (*). EDIT inserts text before the current line if you give I on its own, or follow it with a period (.). If you type an asterisk (*) after I, your text is inserted at the end of the file (that is, before the end-of-file line). Any text that you type is inserted before the line you specified.

To indicate the end of your insertion, press RETURN, type Z, and press RETURN again. For example,

```
I 468
The little fishes of the sea,
They sent an answer back to me.
Z
```

inserts the two lines of text before line 468.

If you omit the line number from the command, EDIT inserts the new material before the current line. For example,

```
F/corkscrew/; I
He said, "I'll go and wake them, if..."
Z
```

This multiple command finds the line containing 'corkscrew' (which then becomes the current line) and inserts the specified new line.

After an I command containing a line number, the current line is the line of that number; otherwise, the current line is unchanged.

To insert material at the end of the file, type I*.

To save you typing, EDIT provides the R (Replace) command, the exact equivalent of typing DI (D for Delete followed by I for Insert). For example,

```
R19 26
In winter when the fields are white
Z
```

deletes lines 19 to 26 inclusive, then inserts the new material before line 27, which becomes the current line.

4.1.2.8 Command Repetition

You can also use individual repeat counts as shown in the examples for N and D above with many EDIT commands. In addition, you can repeat a collection of commands by forming them into a command group using parentheses as follows:

```
6(F P//; D)
```

deletes the next six blank lines in the source. Command groups may not extend over more than one line of command input.

4.1.3 Leaving EDIT

To end a EDIT session, you use the command W (for Windup). EDIT 'winds through' to the end of the source, copying it to the destination, and exits. Unless you specify a TO file, EDIT renames the temporary output file as the FROM filename.

EDIT can accept commands from a number of command sources. In the simplest case, EDIT accepts commands directly from the terminal (that is, from the keyboard); this is called the **primary command level**. EDIT can, however, accept commands from other sources, for example, **command files** or WITH files.

You can call command files from within EDIT, and further command files from within command files, with the C command, so that each nested command file becomes a separate command level. EDIT stops executing the commands in the command file when it comes to the end of the command file, or when it finds a Q. When EDIT receives a Q command in a command file, or it comes to the end of the file, it immediately stops executing commands from that file, and reverts to the the previous command level. If EDIT finds a Q command in a nested command file, it returns to executing commands in the command file at the level above. If you stop editing at the primary command level, by typing Q, or if EDIT finds a Q in a WITH file, then EDIT winds up and exits in the same way as it does with W.

The command STOP terminates EDIT without any further processing. In particular, EDIT does not write out any pending lines of output still in memory so that the destination file is incomplete. If you only specify the FROM argument, EDIT does not overwrite the source file with the (incomplete) edited file. You should only use STOP if you do not need the output from the EDIT run.

EDIT writes a temporary backup to :T/ED-BACKUP when you exit with the W or Q commands. This backup file remains until you exit from EDIT with these commands again, whereupon EDIT overwrites the file with a new backup. If you use the STOP command, EDIT does not write to this file.

4.1.4 A Combined Example: Pulling It All Together

You can meet most simple editing requirements with the commands already described. This section presents an example that uses several commands. The text in italics following the editing commands in the example is a comment. You are not meant to type these comments; EDIT does not allow comments in command lines.

To make it easier for you to follow what is happening, we have included this file as "Edit_Sample" on your accompanying disk.

Take the following source text (with line numbers):

```
1  Tweedledee and Tweedledum
2  agreed to a battle,
3  For Tweedledum said Tweedledee
4  ad spoiled his nice new rattle.
5
6  As black as a tar barrel
7  Which frightened both the heroes so
8  They quite forgot their quorell
```

Execute these EDIT commands:

```
M1; E/dum/dee/; E/dee/dum/  the order of the
                            E commands matters!
N; E/a/A/; B /a /have /     now at line 2
F B/ad/; B//H/              H at line start
F P//; N; I                 before line after blank one
Just then flew down a monstrous crow,
Z
M6; 2(A L//,/; N)           commas at end of lines
F/quore/; E/quorell/quarrel./
                            F is in fact redundant
W                           Windup
```

The following text (with new line numbers) is the result.

```
1  Tweedledum and Tweedledee
2  Agreed to have a battle,
3  For Tweedledum said Tweedledee
4  Had spoiled his nice new rattle.
5
6  Just then flew down a monstrous crow,
7  As black as a tar barrel,
8  Which frightened both the heroes so,
9  They quite forgot their quarrel.
```

Note: If you experiment with editing this source file, you'll find that you don't have to use the commands in the example above. For instance, on the second line, you could use the following command:

```
E/a/have a/
```

to produce the same result.

4.2 A Complete Specification of EDIT

After reading the first part of this chapter on the basic features of EDIT, you should be able to use the editor in a simple way. The rest of this chapter is a reference section that provides a full specification of all the features of EDIT. You may need to consult this section if you have any problems when editing or if you want to use EDIT in a more sophisticated way.

The features described in this section are as follows:

- Command syntax
- Control of Command, Input, and Output Files
- Processing EDIT
- Functional Groupings of EDIT Commands
- Line Windows
- String Operations on the Current Line
- Miscellaneous Current Line Commands
- Inspecting Parts of the Source: The Type Commands
- Control of Command, Input, and Output Files
- Loops
- Global Operations
- Displaying the Program State
- Terminating an EDIT Run
- Current Line Verification
- Miscellaneous Commands
- Abandoning Interactive Editing

4.2.1 Command Syntax

EDIT commands consist of a command name followed by zero or more arguments. One or more space characters may optionally appear between a command name and the first argument, between non-string arguments, and between commands. A space character is only necessary in these places to separate successive items otherwise treated as one (for example, two numbers).

EDIT understands that a command is finished in any of the following ways: when you press RETURN; when EDIT reaches the end of the command arguments; or when EDIT reads a semicolon (;), or closing parenthesis ( )), that you have typed.

You use parentheses to delimit command groups.

To separate commands that appear on the same line of input, you type a semicolon. This is only strictly necessary in cases of ambiguity where a command has a variable number of arguments. EDIT always tries to read the longest possible command.

Except where they appear as part of a character string, EDIT thinks of upper and lower case letters as the same.

4.2.1.1 Command Names

A command name is either a sequence of letters or a single special character (for example, #). An alphabetic command name ends with any non-letter; only the first four letters of the name are significant. One or more spaces may appear between command names and their arguments; EDIT requires at least one space when an argument starting with a letter follows an alphabetic name.

4.2.1.2 Arguments

The following sections describe the six different types of argument you can use with EDIT commands:

- strings
- qualified strings
- search expressions
- numbers
- switch values
- command groups

4.2.1.3 Strings

A string is a sequence of up to 80 characters enclosed in delimiters. You may use an empty (null) string. (A null string is exactly what it sounds like: a non-string, that is, delimiters enclosing nothing, for example, //.) The character that you decide to use to delimit a particular string may not appear in the string. The terminating delimiter may be omitted if it is immediately followed by the end of the command line.

The following characters are available for use as delimiters:

```
/  .  +  -  ,  ?  :  *
```

that is, common English punctuation characters (except ;) and the four arithmetic operators.

Here are some examples of strings:

```
/A/
*Menai Bridge*
??
+String with final delimiter omitted
```

4.2.1.4 Multiple Strings

Commands that take two string arguments use the same delimiter for both and do not double it between the arguments. An example is the A command:

```
A /King/The Red /
```

For all such commands the second string specifies replacement text. If you omit the second string, EDIT uses the null string. If you do this with the A and B command, then nothing happens because you have asked EDIT to insert nothing after or before the first string. However, if you omit the second string after an E command, EDIT deletes the first string.

4.2.1.5 Qualified Strings

Commands that search for contexts, either in the current line or scanning through the source, specify the context with qualified strings. A qualified string is a string preceded by zero or more qualifiers. The qualifiers are single letters. They may appear in any order. For example,

BU/Abc/

Spaces may not appear between the qualifiers. You may finish a list of qualifiers with any delimiter character. The available qualifiers are B (Beginning), E (End), L (Left or Last), P (Precisely), and U (Uppercase).

4.2.1.6 Search Expressions

Commands that search for a particular line in the source take a search expression as an argument. A search expression is a single qualified string. For example,

```
F B/Tweedle/
```

tells EDIT to look for a line beginning with the string 'Tweedle'.

4.2.1.7 Numbers

A number is a sequence of decimal digits. Line numbers are a special form of number and must always be greater than zero. Wherever a line number appears, the characters '.' and '*' may appear instead. A period represents the current line, and an asterisk represents the last line at the end of the source file. For example,

```
M*
```

instructs EDIT to move to the end of the source file.

4.2.1.8 Switch Values

Commands that alter EDIT switches take a single character as an argument. The character must be either a + or -. For example, in

```
V-
```

the minus sign (-) indicates that EDIT should turn off the verification. If you then type V+, EDIT turns the verification on again. In this case, you can consider + as 'on' and - as 'off'.

4.2.1.9 Command Groups

To make a number of individual EDIT commands into a command group, you can enclose them in parentheses. For example, the following line:

```
(f/Walrus/;e/Walrus/Large Marine Mammal/)
```

finds the next occurrence of 'Walrus' and changes it to 'Large Marine Mammal'. Command groups, however, may not span more than one line of input. For instance, if you type a command group that is longer than one line, EDIT only accepts 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
```

Note that it is only necessary to use parentheses when you intend to repeat a command group more that once.

4.2.1.10 Command Repetition

EDIT accepts many commands preceded by an unsigned decimal number to indicate repetition, for example

```
24N
```

If you give a value of zero, then EDIT executes the command indefinitely (or until end-of-file is reached). For example, if you type

```
0(e /dum/dee/;n)
```

EDIT exchanges every occurrence of 'dum' for 'dee' to the end of the file.

You can specify repeat counts for command groups in the same way as for individual commands:

```
12(F/handsome/; E/handsome/hansom/; 3N)
```

4.2.2 Processing EDIT

This section describes what happens when you run EDIT. It gives details about where input comes from and where the output goes, what should appear on your screen, and what should eventually appear in your file after you have run EDIT.

4.2.2.1 Prompts

When EDIT is being run interactively, that is, with both the command file connected to the keyboard and the verification file connected to a window, it displays a prompt when it is ready to read a new line of commands. Although, if the last command of the previous line caused verification output, EDIT does not return a prompt.

If you turn the verification switch V on, EDIT verifies the current line in place of a prompt in the following circumstances:

- if it has not already verified the current line,
- if you have made any changes to the line since it was last verified, *or*
- if you have changed the position of the operational window.

Otherwise, when EDIT does not verify the current line, it displays a colon character (:) to indicate that it is ready for a new line of commands. This colon is the usual EDIT prompt.

EDIT never gives prompts when you are inserting lines.

4.2.2.2 The Current Line

As EDIT reads lines from the source file and writes them to the destination file, the line that EDIT has in its hand at any time is called the **current line**. Every command that you type refers to the current line. EDIT inserts new lines before the current line. When you start editing with EDIT, the current line is the first line of the source.

4.2.2.3 Line Numbers

EDIT identifies each line in the source by a unique line number. This is not part of the information stored in the file. EDIT computes these numbers by counting the lines as it reads them. EDIT does not assign line numbers to any new lines that you insert into the source.

EDIT distinguishes between original and non-original lines. Original lines are source lines that have not been split or inserted; non-original lines are split lines and inserted lines. Commands that take line numbers as arguments may only refer to original lines. EDIT moves forward, or backward up to a set limit, according to whether the line number you type is greater or less than the current line number. EDIT passes over or deletes (if appropriate) non-original lines in searches for a given original line.

When you type a period (.) instead of a line number, EDIT always uses the current line whether original or non-original. (For an example of its use, see Section 4.1.2.6, Deleting Whole Lines.)

You can renumber lines with the '=' command. This ensures that all lines following the current line are original. Type

```
=15
```

to number the current line as 15, the next line 16, the next 17, and so on to the end of the file. This is how you allocate line numbers to non-original lines. If you do not qualify the = command with a number, EDIT displays the message:

```
Number expected after =
```

4.2.2.4 Qualified Strings

To specify contexts for EDIT searches, you can use qualified strings. EDIT accepts the null string and always matches it at the initial search position, which is the beginning of the line except as specified below. In the absence of any qualifiers, EDIT may find the given string anywhere in a line. Qualifiers specify additional conditions for the context. EDIT recognizes five qualifiers B, E, L, P, and U as follows:

B

where the string must be at the Beginning of the line. This qualifier may not appear with E, L, or P.

E

where the string must be at the End of the line. This qualifier may not appear with B, L, or P. If E appears with the null string, it matches with the end of the line. (That is, look for nothing at the end of a line.)

L

where the search for the string is to take place Leftwards from the end of the line instead of rightwards from the beginning. If there is more than one occurrence of the string in a line, this qualifier makes sure that the Last one is found instead of the first. L may not appear with B, E, or P. If L appears with the null string, it matches with the end of the line. (That is, look leftwards from the end of the line for an occurrence of nothing.)

P

where the line must match the string Precisely and must contain no other characters. P must not appear with B, E, or L. If P appears with a null string, it matches with an empty line.

U

where the string match is to take place whether or not upper or lower case is used. (That is, as though you translated both the string and the line into Uppercase letters before comparing them.) For example, when you specify U, the following string

```
/TWEEDledum/
```

should match a line containing

TweedleDUM

as well as any other combination in upper or lower case.

4.2.2.5 Output Processing

EDIT does not write lines read in a forward direction to the destination file immediately, but instead it adds them to an output queue in main memory. When EDIT has used up the memory available for such lines, it writes out the lines at the head of the queue as necessary. Until EDIT has actually written out a line to the destination file, you can move back and make it the current line again.

You can also send portions of the output to destination files other than TO. When you select an alternative destination file, EDIT writes out the queue of lines for the current output file.

4.2.2.6 End-of-File Handling

When EDIT reaches the end of a source file, a dummy end-of-file line becomes current. This end-of-file 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.

When the end-of-file line is current, commands to make changes to the current line, and commands to move forward, produce an error. Although, if you contain these kinds of commands within an infinitely repeating group, EDIT does not give an error on reaching the end-of-file line. The E (Exchange) command is an example of a command to make changes to the curre nt line. The N (Next) command is an example of a command to move forward.

4.2.3 Functional Groupings of EDIT Commands

This section contains descriptions of all EDIT commands split up by function. A summary and an alphabetic list of commands appear later.

The following descriptions use slashes (/) to indicate delimiter characters (that is, characters that enclose strings). Command names appear in upper case; argument types appear in lower case as follows:

Notation	Description
a, b	line numbers (or . or *)
cg	command group
m, n	numbers
q	qualifier list (possibly empty)
se	search expression
s, t	strings of arbitrary characters
sw	switch value (+ or -)
/	string delimiter

Table 4.1 Notation for Command Descriptions

Note: Command descriptions that appear in the rest of this manual with the above notation show the SYNTAX of the command; they are not examples of what you actually type. Examples always appear as follows in

```
this typeface.
```

4.2.3.1 Selection of a Current Line

These commands have no function other than to select a new current line. EDIT adds lines that it has passed in a forward direction to the destination output queue (for further details on the output queue, see Section 4.1, Introducing EDIT). EDIT queues up lines that it has passed in a backward direction ready for subsequent reprocessing in a forward direction. M takes a line number, period, or asterisk. So, using the command notation described above, the correct syntax for M is as follows:

```
Ma
```

where Ma moves forward or backward to line 'a' in the source. Only original lines can be accessed by line number.

```
M+
```

makes the last line actually read from the file current line. M+ moves through all the lines currently held in memory until the last one is reached.

```
M-
```

makes the last line on the output queue current. This is like saying to EDIT: "move back as far as you can."

```
N
```

moves forward to the next line in the source. When the current line is the last line of the source, executing an N command does not create an error. EDIT increases the line number by adding one to it and creates a special end-of-file line. However, if you try to use an N command when you are already at the end of the source file, EDIT returns an error.

```
P
```

moves back to the previous line. You can move more than one line back by either repeating P, or giving a number before it. The number that you give should be equal to the number of lines you want to move back.

The syntax for the F (Find) command is

```
F se
```

So, F finds the line you specify with the search expression 'se'. The search starts at the current line and moves forward through the source. The search starts at the current line in order to cover the case where the current line has been reached as a side effect of previous commands - such as line deletion. An F command with no argument searches using the last executed search expression.

The syntax for the BF (Backwards Find) command is

```
BF se
```

BF behaves like F except that it starts at the current line and moves backward until it finds a line that matches its search expression.

4.2.3.2 Line Insertion and Deletion

Commands may select a new current line as a side effect of their main function. Those followed by in-line insertion material must be the last command on a line. The insertion material is on successive lines terminated by Z on a line by itself. You can use the Z command to change the terminator. EDIT recognizes the terminator you give in either upper or lower case. For example, using the same notation,

```
Ia
<insertion material, as many
lines as necessary>
Z
```

inserts the insertion material before 'a'. Remember that 'a' can be a specific line number, a period (representing the current line), or an asterisk (representing the last line of the source file). If you omit a, EDIT inserts the material before the current line; otherwise, line a becomes the current line.

```
I /s/
```

inserts the contents of the file s (remember, 's' means any string) before the current line.

```
Ra b
<replacement material>
Z
Ra b /s/
```

The R command is equivalent to D followed by I. The second line number must be greater than or equal to the first. You may omit the second number if you want to replace just the one line (that is, if b=a). You may omit both numbers if you want to replace the current line. The line following line b becomes the new current line.

The syntax for the D (Delete) command is as follows:

```
Da b
```

So, D deletes all lines from a to b inclusive. You may omit the second line number if you want to delete just the one line (that is, if b=a). You may omit both numbers if you want to delete the current line. The line following line b becomes the new current line.

The syntax of the DF (Delete Find) command is

```
DF se
```

The command DF (Delete Find) tells EDIT to delete successive lines from the source until it finds a line matching the search expression. This line then becomes the new current line. A DF command with no argument searches (deleting as it goes) using the last search expression you typed.

4.2.4 Line Windows

EDIT usually acts on a complete current line. However, you can define parts of the line where EDIT can execute your subsequent commands. These parts of lines are called **line windows**. This section describes the commands you use to define a window.

4.2.4.1 The Operational Window

EDIT usually scans all the characters in a line when looking for a given string. However it is possible to specify a 'line window', so that the scan for a character starts at the beginning of the window, and not the start of the line. In all the descriptions of EDIT context commands, "the beginning of the line" always means "the beginning of the operational window".

Whenever EDIT verifies a current line, it indicates the position of the operational window by displaying a '>' character directly beneath the line. For example in the following

```
26.
This is line 26 this is.
      >
```

the operational window contains the characters to the right of the pointer: "line 26 this is.". EDIT omits the indicator if it is at the start of the line.

The left edge of the window is also called the **character pointer** in this context, and the following commands are available for moving it:

```
>
```

moves the pointer one character to the right.

```
<
```

moves the pointer one character to the left.

PR

Pointer Reset sets the pointer to the start of the line.

The syntax for the PA (Point After) command is

PA q/s/

Point After: sets the pointer so that the first character in the window is the first character following the string s. For example,

```
PA L//
```

moves the pointer to the end of the line.

The syntax for the PB (Point Before) command is

PB q/s/

Point Before is the same as PA, but includes the string itself in the window.

4.2.4.2 Single Character Operations on the Current Line

The following two commands move the character pointer one place to the right after forcing the first letter into either upper or lower case. If the first character is not a letter, or is already in the required case, these commands are equivalent to >.

The command

```
$
```

forces lower case (Dollar for Down).

The command

```
%
```

forces upper case (Percent for uP).

The '_' (underscore) command changes the first character in the window into a space character, then moves the character pointer one place to the right.

The command

```
#
```

deletes the first character in the window. The remainder of the window moves one character to the left, leaving the character pointer pointing at the next character in the line. The command is exactly equivalent to

```
E/s//
```

where s is the first character in the window. To repeat the effect, you by specify a number before the '#' command. If the value is n, for example, then the repeated command is equivalent to the single command

```
E/s//
```

where s is the first n characters in the window or the whole of the contents of the window, whichever is the shorter. Consider the following example:

```
5#
```

deletes the next five characters in the window. If you type a number equal to or greater than the number of characters in the window, EDIT deletes the contents of the entire window. EDIT treats a sequence of '#' commands in the same way as a single, repeated '#' command. So, ##### is the same as typing a single #, pressing RETURN after each single #, five times.

You can use a combination of '>' '%' '$' '_' and '#' commands to edit a line character by character, the commands appearing under the characters they affect. The following text and commands illustrate this:

```
o Oysters,, Come ANDDWALK with    us
%>$$$$$$$#>>$$$$$$$$_$$$$$$$$$$###
```

The commands in the example above change the line to

```
O oysters, come and walk with us
```

leaving the character pointer immediately before the word 'us'.

4.2.5 String Operations on the Current Line

To specify which part of the current line to qualify, you can either alter the basic string or point to a variant, as described in the next two sections.

4.2.5.1 Basic String Alterations

Three similar commands are available for altering parts of the current line. The A, B and E commands insert their second (string) argument After, Before, or in Exchange for their first argument respectively. You may qualify the first argument. If the current line were

```
The Carpenter beseech
```

then the commands

```
E U/carpenter/Walrus/        Exchange
B/bese/did /                 Insert string before
A L//;/                      Insert string after
```

would change the line to

```
The Walrus did beseech;
```

4.2.5.2 The Null String

You can use the null, or empty string (//) after any string command. If you use the null string as the second string in an E command, EDIT removes the first string from the line. Provided EDIT finds the first string, an A or B command with a null second string does nothing; otherwise, EDIT returns an error. A null first string in any of the three commands matches at the initial search position. The initial search position is the current character position (initially the beginning of the line) unless either of the E or L qualifiers is present, in which case the initial position is the end of the line. For example,

```
A//carpenter/
```

puts the text carpenter After nothing, that is, at the beginning of the line. Whereas

```
A L//carpenter
```

puts carpenter at the end of the line After the Last nothing.

4.2.5.3 Pointing Variant

The AP (insert After and Point), BP (insert Before and Point), and EP (Exchange and Point) commands take two strings as arguments and act exactly like A, B, and E. However, AP, BP, and EP have an additional feature: when the operation is complete, the character pointer is left pointing to the first character following both text strings. So, using the same command syntax notation,

AP/s/t/

is equivalent to

A/s/t/; PA/st/

while

BP/s/t/

is equivalent to

B/s/t/; PA/ts/

and

```
2EP U/tweadle/Tweedle/
```

would change

```
tweadledum and TWEADLEdee
```

into

```
Tweedledum and Tweedledee
```

leaving the character pointer just before dee.

4.2.5.4 Deleting Parts of the Current Line

You use the commands DTA (Delete Till After) and DTB (Delete Till Before) to delete from the beginning of the line (or character pointer) to a specified string. To delete from a given context until the end of the line, you use the commands DFA (Delete From After) and DFB (Delete From Before). If the current line were

```
All the King's horses and all the King's men
```

then the command

```
DTB L/King's/
```

would change it to

```
King's men
```

while

```
DTA/horses /
```

would change it to

```
and all the King's men
```

4.2.6 Miscellaneous Current Line Commands

This section includes some further commands that explain how to repeat commands involving strings, how to split the current line, and how to join lines together.

Whenever EDIT executes a string alteration command (for example, A, B, or E), it becomes the **current string alteration command**. To repeat the current string alteration command, you can type a single quote ('). The ' command has no arguments. It takes its arguments from the last A, B, or E command.

WARNING: Unexpected effects occur if you use sequences such as

```
E/castle/knight/; 4('; E/pawn/queen/)
```

The second and subsequent executions of the ' command refer to a different command than the first. The above example would exchange castle and knight twice and exchange pawn and queen seven times instead of exchanging castle and knight once and then four times exchanging castle and knight and pawn and queen.

4.2.6.1 Splitting and Joining Lines

EDIT is primarily a line editor. Most EDIT editing commands do not operate over line boundaries, but this section describes commands for splitting a line into more than one line and for joining together two or more successive lines.

To split a line before a specified context, you use the SB command. The syntax for the SB command is

SB q/s/

SB takes an optional qualifier represented here by q, and a string /s/. SB Splits the current line Before the context you specify with the qualifier and string. EDIT sends the first part of the line to the output and makes the remainder into a new, non-original current line.

To split a line after a specified context, you use the SA command. The syntax for SA is

SA q/s/

SA takes an optional qualifier and a string (q and /s/). SA Splits the current line After the context you specify with the qualifier and string.

To concatenate a line, you use the CL command. The syntax for CL is

CL/s/

CL takes an optional string that is represented here by /s/. CL or Concatenate Line forms a new current line by concatenating the current line, the string you specified and the next line from the source, in that order. If the string is a null string, you may type the command CL without specifying a string.

For an example of splitting and joining lines, look at the text

```
Humpty Dumpty sat on a wall; Humpty
Dumpty had a
great fall.
```

The old verse appears disjointed; the lines need to be balanced. If you make the first line the current line, the commands

```
SA /; /; 2CL/ /
```

change the line into

```
Humpty Dumpty sat on a wall;
```

leaving

```
Humpty Dumpty had a great fall.
```

as the new current line.

4.2.7 Inspecting Parts of the Source: the Type Commands

The following commands all tell EDIT to advance through the source, sending the lines it passes to the verification file as well as to the normal output (where relevant). Because these commands are most frequently used interactively (that is, with verification to the screen), they are known as the 'type' commands. They have this name because you can use them to 'type' out the lines you specify on the screen. This does not however mean that you cannot use them to send output to a file. After EDIT has executed one of these commands, the last line it 'typed' (that is, displayed) becomes the new current line.

The syntax for the T (Type) command is

Tn

Tn types n lines. If you omit n, typing continues until the end of the source. However, you can always interrupt the typing with CTRL-C.

Note: Throughout this manual when you see a hyphen between two keys, you press them at the same time. So CTRL-C means to hold down the CTRL key while you type C.

When you use the T command, the first line EDIT types is the current line, so that, for example,

```
F /It's my own invention/; T6
```

types six lines starting with the one containing 'It's my own invention'. (Note that to find the correct line, you must type the 'I' in 'It's' in upper case.)

The command

TP

types the lines in the output queue. Thus, TP (Type Previous) is equivalent to EDIT executing M- followed by typing until it reaches the last line it actually read from the source.

The command

TN

types until EDIT has changed all the lines in the output queue. (For more information on the output queue, see Section 4.1, Introducing EDIT) So, a TN (Type Next) command types N lines, where N was the number specified as the P option. (To find out more about the P option, refer to Section 4.1.1, Calling EDIT). The advantage of the TN command is that everything visible during the typing operation is available in memory to P and BF commands.

The syntax for the TL (Type with Line numbers) command is as follows:

TLn

TLn types n lines as for T, but with line numbers added. Inserted and split lines do not have line numbers, EDIT displays a '+ + + +' instead. For example,

```
20   O oysters, come and walk with us
++++ and then we'll have some tea
```

The original line starting with 'O oysters' has a line number. The non-original line, inserted after line 20, starts with + + + +. (Remember that you can use the = command to renumber non-original lines.)

4.2.8 Control of Command, Input and Output Files

EDIT uses four types of files:

- command
- input
- output
- verification

Once you have entered EDIT, you cannot change the verification file with a command. (To find out more about the verification file, see Section 4.1.1, Calling EDIT.) The following sections describe commands that can change the command, input, and output files that you set up when you enter EDIT.

4.2.8.1 Command Files

When you enter EDIT, it reads commands from the terminal or the file that you specify as WITH. To read commands from another file, you can use the C command. The syntax for the command is

C .s.

where the string 's' represents a filename. As AmigaDOS uses the slash symbol (/) to separate filenames, you should use periods (.), or some other symbol, to delimit the filename. A symbol found in a string should not be used as a delimiter. When EDIT has finished all the commands in the file (or you give a Q command), it closes the file and control reverts to the command following the C command. For example, the command

```
C .:T/XYZ.
```

reads and executes commands from the file :T/XYZ

4.2.8.2 Input Files

To insert the entire contents of a file at a specific point in the source, you use the I and R commands. These commands are described in Section 4.1.2.7 earlier in this chapter.

Section 4.1.1 described how to call EDIT. In that section, the source file was referred to as the FROM file. However, you can also associate the FROM file with other files, using the command FROM. The FROM command has the following form:

FROM .s.

where the string 's' is a filename. A FROM command with no argument re-selects the original source file.

When EDIT executes a FROM command, the current line remains current; however, the next line comes from the new source.

EDIT does not close a source file when the file ceases to be current; you can read further lines from the source file by re-selecting it later.

To close an output file that you opened in EDIT, and that subsequently you want to open for input (or the other way round), you must use the CF (Close File) command. The CF command has the following form:

CF .s.

where the string 's' represents a filename. When you end an EDIT session, EDIT closes automatically all the files you opened in EDIT.

Note: Any time you open a file, EDIT starts at the beginning of that file. If you close a file with CF, EDIT starts on the first line of that file if you re-open it, and not at the line it was on when you closed the file.

An example of the use of the FROM command to merge lines from two files follows:

Command	Action
M10	*Pass lines 1-9 from the FROM (source) file*
FROM .XYZ.	*Select new input, line 10 remains current*
M6	*Pass line 10 from FROM, lines 1-5 from XYZ*
FROM	*Re-select FROM*
M14	*Pass line 6 from XYZ, lines 11-13 from FROM*
FROM .XYZ.	*Re-select XYZ*
M*	*Pass line 14 from FROM, the rest of XYZ*
FROM	*Re-select FROM*
CF .XYZ.	*Close XYZ*
M*	*Pass the rest of FROM (lines 15 till end-of-file)*

4.2.8.3 Output Files

EDIT usually sends output to the file with filename TO. However, EDIT does not send the output immediately. It keeps a certain number of lines in a queue in main memory as long as possible. These lines are previous current lines or lines that EDIT has passed before reaching the present current line. The number of lines that EDIT can keep depends on the options you specified when you called EDIT. Because EDIT keeps these lines, it has the capability for moving backwards in the source.

To associate the output queue with a file other than that with the filename TO, you can also use the TO command. The TO command has the form

TO .s.

where s is a filename.

When EDIT executes a TO command, it writes out the existing queue of output lines if the output file is switched.

EDIT does not close an output file when it is no longer current. By re-selecting the file, you can add further lines to it. The following example shows how you can split up the source between the main destination TO and an alternate destination XYZ.

Command	Action
M11	*Pass lines 1-10 to TO*
TO.XYZ.	*Switch output file*
M21	*Pass lines 11-20 to XYZ*
TO	
M31	*Pass lines 21-30 to TO*
TO.XYZ.	
M41	*Pass lines 31-40 to XYZ*
TO	

If you want to re-use a file, you must explicitly close it. The command

CF .filename.

closes the file with the filename you specify as the argument.

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 .:T/1.`	*Output to temporary file*
`1000N`	*Advance through source*
`TO`	*Revert to TO*
`CF .:T/1.`	*Close output file :T.1*
`I2000.:T/1.`	*Re-use as input file*

If you use the CF command on files you have finished with, the amount of memory you need is minimized.

4.2.9 Loops

You can type an unsigned decimal number before many commands to indicate repetition, for example,

```
24N
```

You can also specify repeat counts for command groups in the same way as for individual commands, for example,

```
12(F/handsome/; E/handsome/hansom/; 3N)
```

If you give a repeat count of zero (0), the command or command group is repeated indefinitely or until EDIT reaches the end of the source.

4.2.10 Global Operations

Global operations are operations that take place automatically as EDIT scans the source in a forward direction. You can start and stop global operations with special commands, described in the following sections.

WARNING: Be careful when you move backwards through the source not to leave any active or 'enabled' globals. These enabled globals could undo a lot of your work!

4.2.10.1 Setting Global Changes

Three commands, GA, GB, and GE are provided for simple string changes on each line. Their syntax is as follows:

GA q/s/t/
GB q/s/t/
GE q/s/t/

These commands apply an A, B or E command, as appropriate, to any occurrence of string 's' in a new current line. They also apply to the line that is current at the time the command is executed.

G commands do not re-scan their replacement text; for example, the following command

```
GE/Tiger Lily/Tiger Lily/
```

would not loop forever, but would have no visible effect on any line. However, as a result of the 'change', EDIT would verify certain lines.

EDIT applies the global changes to each new current line in the order in which you gave the commands.

4.2.10.2 Cancelling Global Changes

The REWIND command cancels all global operations automatically. You can use the CG (Cancel Global) command to cancel individual commands at any time.

When a global operation is set up by one of the commands GA, GB, or GE, the operation is allocated an identification number which is output to the verification file (for example, G1). The argument for CG is the number of the global operation to be cancelled. If CG is executed with no argument, EDIT cancels all globals.

4.2.10.3 Suspending Global Changes

You can suspend individual global operations, and later resume using them with DG (Disable Global) and EG (Enable Global) commands. These take the global identification number as their argument. If you omit the argument, all globals are turned off or on (disabled or enabled), as appropriate.

4.2.11 Displaying the Program State

Two commands beginning with SH (for SHow) output nformation about the state of EDIT to the verification file.

The command SHD (SHow Data) takes the form

SHD

and displays saved information values, such as the last search expression.

The command SHG (SHow Globals) takes the form

SHG

and displays the current global commands, together with their identification numbers. It also gives the number of times each global search expression matches.

4.2.12 Terminating an EDIT Run

To 'wind through' the rest of the source, you use the W command (Windup). Note that W is illegal if output is not currently directed to TO. EDIT exits when it has reached the end of the source, closed all the files, and relinquished the memory. Reaching the end of the highest level command file has the same effect as W. If you call EDIT specifying only the FROM filename, EDIT renames the temporary output file it created with the same name as the original (that is, the FROM filename), while it renames the original information as the file :T/EDIT-BACKUP. This backup file is, of course, only available until the next time EDIT is run.

The STOP command stops EDIT immediately. No further input or output is attempted. In particular, the STOP command stops EDIT from overwriting the original source file. Typing STOP ensures that no change is made to the input information.

The Q command stops EDIT from executing the current command file (EDIT initially accepts commands from the keyboard, but you can specify a command file with the WITH keyword or with the C command) and makes it revert to the previous one. A Q at the outermost level does the same as a W.

4.2.13 Current Line Verification

The following circumstances can cause automatic verification to occur:

- When you type a new line of commands for a current line that EDIT has not verified since it made the line current, or 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.

In the first two cases, the verification only occurs if the V switch is on. The command

```
V sw
```

changes the setting of the V switch. It is set ON (V+) if the initial state of EDIT is interactive (commands and verifications both connected to a terminal), and to OFF (V-) otherwise.

To explicitly request verification of the current line, you use the following command

```
?
```

This command verifies the current line. It is performed automatically if the V switch is on and the information in the line has been changed. The verification consists of the line number (or + + + + if the line is not original), with the text on the next line.

An alternate form of verification, useful for lines containing non-printing characters, is provided by the command

```
!
```

The ! command verifies the current line with character indicators. EDIT produces two lines of verification. The first is the current line in which EDIT replaces all the 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.

The following example uses the ? and ! commands. To verify the current line, you use the ? command. If, for instance, the following appears when you use the ? command:

```
?
1.
The Walrus and the ??
```

then you might try to use the E command to exchange '??' for 'Carpenter'. However, EDIT may not recognize the text it displayed with '??' as two question marks if the '??' characters correspond to two non-graphic characters. To find out what really is there, you use the ! command as follows:

```
!
1.
The Walrus and the 11
-   -              44
```

To correct the line, you can use the character pointer and # command to delete the spurious characters before inserting the correct text. (For further details on using the character pointer and # command, see Section 4.2.4, Line Windows.)

4.2.14 Miscellaneous Commands

This section describes all those commands that do not fit neatly into any of the previous categories. It describes how to change a termination character, turn trailing spaces off, renumber lines, and rewind the source file.

To change the terminator for text insertion, you use the Z command. The Z command has the following form

Z /s/

where /s/ represents a string. The string may be of any length up to 16 characters. The string is matched in either case. In effect, the search for the terminator is done using the qualifiers PU. The initial terminator string is Z.

To turn trailing spaces on or off, you use the TR (TRailing spaces) command. The TR command takes the following form

TR sw

where sw represents a switch (+ for ON; - for OFF). EDIT usually suppresses all trailing spaces. TR+ allows trailing spaces to remain on both input and output lines.

To renumber the source lines, you use the = command. The = command takes the form

=n

where n represents a number. The command =n sets the current line number to n. If you then move to the lines below the current line, EDIT renumbers all the following original and non-original lines. Although, if you move back to previous lines after using the = command, EDIT marks all the previous lines in the output queue as non-original. When you rewind the source file, EDIT renumbers all the

lines in the file - original, non-original, and those previously re-numbered with the = command.

To rewind the source file, you use the REWIND command. For example,

```
REWIND
```

This command rewinds the input file so that line 1 is the current line again. First EDIT scans the rest of the source (for globals, and so forth). Then it writes the lines to the destination, which is then closed and re-opened as a new source. It closes the original source using a temporary file as a destination. Any globals that you specify are cancelled. EDIT does not necessarily require you to type the complete word (that is, REWIND). To move to the beginning, you can type any of the following: REWI, REWIN, or REWIND.

4.2.15 Abandoning Interactive Editing

To abandon most commands that read text, you press CTRL-C. In particular, if you realize that a search expression has been mistyped, then CTRL-C stops the search. Similarly the T command types to the end of the source, but CTRL-C abandons this action.

After you press CTRL-C, EDIT responds with the message

```
*** BREAK
```

and returns to reading commands. The current line does, of course, depend on exactly when you pressed CTRL-C.

Quick Reference Card

This list uses the following abbreviations

Notation	Description
qs	Qualified string
t	String
n	Line number, or . or * (current and last line)
sw	+ or - (on or off)

Character Pointer Commands (Line Window Commands)

Command	Action
<	Move character pointer left
>	Move character pointer right
#	Delete character at pointer
$	Lower case character at pointer
%	Upper case character at pointer
_	Turn character at pointer to space
PA qs	Move character pointer to after qs
PB qs	Move character pointer to before qs
PR	Reset character pointer to start of line

Positioning Commands

Command	Action
M n	Move to line n
M +	Move to highest line in memory
M -	Move to lowest line in memory
N	Next line
P	Previous line
REWIND	Rewind input file

Search Commands

Command	Action
F qs	Find string qs
BF qs	Same as F, but move backwards through file
DF qs	Same as F, but delete lines as they are passed

Text Verification

Command	Action
?	Verify current line
!	Verify with character indicators
T	Type to end of file
T n	Type n lines
TL n	Type n lines with line numbers
TN	Type until buffer changed
TP	M-, then type to last line in buffer
V sw	Set verification on or off

Operations on the Current Line

Command	Action
A qs t	Place string t after qs
AP qs t	Same as A, but move character pointer
B qs t	Place string t before qs
BP qs t	Same as B, but move character pointer
CL t	Concatenate current line, string t and next line
D	Delete current line
DFA qs	Delete from after qs to end of line
DFB qs	Delete from before qs to end of line
DTA qs	Delete from start of line to after qs
DTB qs	Delete from start of line to before qs
E qs t	Exchange string qs with string t
EP qs t	Same as E, but move character pointer
I	Insert material from terminal before line
I t	Insert from file t
R	Replace material from terminal
R t	Replace material from file t
SA qs	Split line after qs
SB qs	Split line before qs

Globals

Command	Action
GA qs t	Globally place t after qs
GB qs t	Globally place s before qs
GE qs t	Globally exchange qs for t
CG n	Cancel global n (all if n omitted)
DG n	Disable global n (all if n omitted)
EG n	Enable global n (all if n omitted)
SHG	Display info on globals used

Input/Output Manipulation

Command	Action
FROM	Take source from original
FROM t	Take source from file t
TO	Revert to original destination
TO t	Place output lines in file t
CF t	Close file t

Other Commands

Command	Action
'	Repeat previous A, B or E command
= n	Set line number to n
C t	Take commands from file t
H n	Set halt at line n. If n=* then halt and unset h
Q	Exit from command level; windup if at level 1
SHD	Show data
STOP	Stop
TR sw	Set/unset trailing space removal
W	Windup
Z t	Set input terminator to string t

Appendix A: Error Codes and Messages

The error messages that appear on the screen when you use the FAULT or WHY command fall into two general categories:

1. user errors

2. programmer errors.

This appendix gives the probable cause and a suggestion for recovery for each of these error codes. The codes appear in numerical order within their category.

User Errors

103: insufficient free store

Probable cause:

You don't have enough physical memory on the Amiga to carry this operation out.

Recovery suggestion:

First, try to stop some of the applications that are running that you don't need. For example, close any unnecessary windows. Otherwise, buy more memory. Stop some of the tasks that are less important to you and re-issue the command. It may be that you have enough memory, but it has become 'fragmented'; rebooting may help.

104: task table full

Probable cause:

Limited to 20 CLI tasks, or equivalent.

120: argument line invalid or too long

Probable cause:

Your argument for this command is incorrect or contains too many options.

Recovery suggestion:

Consult the command specifications in Chapter 2 of this manual for the correct argument template.

121: file is not an object module

Probable cause:

Either you misspelled the command name, or this file may not be in loadable file form.

Recovery suggestion:

Either re-type the file name, or make sure that the file is a binary program file. Remember that in order to execute a command sequence the command EXECUTE must be used before the file name.

122: invalid resident library during load

202: object in use

Probable cause:

The file or directory specified is already being used by another application in a manner incompatible with the way you want to use it.

Recovery suggestion:

If another application is writing to a file, then nobody else can read from it. If another application is reading from a file, then nobody else can write to it. If an application is using a directory or reading from a file, then nobody else may delete or rename the file or directory. You must stop the other application using the file or directory and then try again.

203: object already exists

Probable cause:

The object name that you specified is that of an object that already exists.

Recovery suggestion:

You must first delete the directory or file if you really want to re-use that name.

204: directory not found

205: object not found

Probable cause:

AmigaDOS cannot find the device or file you specified. You have probably made a typographical or spelling error.

Recovery suggestion:

Check device names and file names for correct spellings. You also get this error if you attempt to create a file in a directory that does not exist.

206: invalid window

Probable cause:

You have either made the dimensions too big or too small, or you have failed to define an entire window. (For example, you must not forget the final slash.) You can also get this error from NEWCLI if you supply a device name that is not a window.

Recovery suggestion:

You should re-specify the window.

210: invalid stream component name

Probable cause:

You have included an invalid character in the filename you have specified, or the filename is too long. Each file or directory must be less than 30 characters long. A filename cannot contain control characters.

212: object not of required type

Probable cause:

Maybe you've tried to do an operation that requires a filename and you gave it a directory name or *vice versa*. For example, you might have given the command TYPE dir, where 'dir' is a directory. AmigaDOS doesn't allow you to display a directory, only files.

Recovery suggestion:

Check on the command usage in Chapter 2 of the *AmigaDOS User's Manual*

213: disk not validated

Probable cause:

Either you just inserted a disk and the disk validation process is in progress, or it may be a bad disk.

Recovery suggestion:

Wait for the disk validation process to finish - it normally only takes less than a minute. If AmigaDOS cannot validate the disk because it is bad, then the disk remains unvalidated. In this case, you can only read from the disk and you must copy your information onto another disk.

214: disk write-protected

Probable cause:

This disk is write-protected. The Amiga cannot write over information that is already on the disk. You can only read information from this disk. You cannot store any information of your own here.

Recovery suggestion:

Save your information on a disk that is not write-protected, or change the write-protect tab on the disk.

215: rename across devices attempted

Probable cause:

RENAME only changes a filename on the same device, although you can use it to rename a file from one directory into another on the same device.

Recovery suggestion:

Copy the file to the object device and delete it from the source device.

216: directory not empty

Probable cause:

You cannot delete a directory unless it is empty.

Recovery suggestion:

Delete the contents of the directory. Study the command specification for DELETE in Chapter 2 of this manual.

218: device not mounted

Probable cause:

The word 'mounted' here means "inserted into the drive"; either you've made a typographical error, or the disk with the desired name isn't mounted.

Recovery suggestion:

Check the spelling of the devices, or insert the correct disk.

220: comment too big

Probable cause:

Your filenote has exceeded the maximum number of characters allowed (80).

Recovery suggestion:

Retype the filenote adhering to these limits.

221: disk full

Probable cause:

You do not have sufficient room on the disk to do this operation.

Recovery suggestion:

Use another disk or delete some unnecessary files or directories.

222: file is protected from deletion

Probable cause:

The file or directory has been protected from deletion.

Recovery suggestion:

You either did not mean to delete that file, or you really did mean it. If you really did mean it, you must use the PROTECT command to alter the protection status. Refer to the PROTECT command in the *AmigaDOS User's Manual*. Also use the LIST command to check on what the protections of this particular file or disk.

223: file is protected from writing

Probable cause:

The file or directory has been protected from being overwritten.

Recovery suggestion:

You either did not mean to write to that file, or you really did mean it. If you really did mean it, you must use the PROTECT command to alter the protection status. Refer to the PROTECT command in the *AmigaDOS User's Manual*. Also use the LIST command to check on what the protections of this particular file or disk.

224: file is protected from reading

Probable cause:

The file or directory has been protected from being read.

Recovery suggestion:

You either did not mean to read from that file, or you really did mean it. If you really did mean it, you must use the PROTECT command to alter the protection status. Refer to the PROTECT command in the *AmigaDOS User's Manual*. Also use the LIST command to check on what the protections of this particular file or disk.

225: not a DOS disk

Probable cause:

The disk in the drive is not a formatted DOS disk.

Recovery suggestion:

Place a suitably formatted DOS disk in the drive instead, or else format the disk using the FORMAT command if you don't want any of the information on it.

226: no disk in drive

Probable cause:

You have attempted to read or write to a disk drive where there is no disk.

Recovery suggestion:

Place a suitably formatted DOS disk in the drive.

Programmer Errors

209: packet request type unknown

Probable cause:

You have asked a device handler to attempt an operation it cannot do (for example, the console handler cannot rename anything).

Recovery suggestion:

Check the request code passed to device handlers.

211: invalid object lock

Probable cause:

You have used something that is not a valid lock.

Recovery suggestion:

Check your code so that you only pass valid locks to AmigaDOS calls that expect locks.

219: seek error

Probable cause:

You have attempted to call SEEK with invalid arguments.

Recovery suggestion:

Make sure that you only SEEK within the file. You cannot SEEK to outside the bounds of the file.

232: no more entries in directory

Probable cause:

There are no more entries in the directory that you are examining.

Recovery suggestion:

This error code indicates that the AmigaDOS call EXNEXT has no more entries in the directory you are examining to hand back to you. Stop calling EXNEXT.

Glossary

Arguments
Additional information supplied to **commands**.

Character pointer
Pointer to the left edge of a line window in EDIT. You use it to define the part of a line that EDIT may alter.

Character string
Sequence of printable characters.

Command
An instruction you give directly to the computer.

Command Line Interpreter (CLI)
A **process** that decodes user input.

Console handler
See **terminal handler**.

Command template
The method of defining the syntax for each **command**.

Control combination
A combination of the CTRL key and a letter or symbol. The CTRL key is pressed down while the letter or symbol is typed. It appears in the documentation, for example, in the form CTRL-A.

Current cursor position
The position the cursor is currently at.

Current directory
This is either the **root directory** or the last **directory** you set yourself in with the command CD.

Current drive
The disk drive that is inserted and declared to be current. The default is SYS:.

Current line
The line that EDIT has in its hand at any time.

Current string alteration command
An instruction that changes the current string.

Delimiter characters
Characters used at the beginning and end of a **character string**.

Destination file
File being written to.

Device name
Unique name given to a device, eg DF0: - floppy drive 0:.

Directory
A collection of **files.**

Editing commands
Commands input from the keyboard that control an editing session.

Extended mode
Commands appear on the command line and are not executed until you finish th e command line.

File
A collection of related data.

Filename
A name given to a **file** for identification purposes.

Immediate mode
Commands are executed immediately.

Keyword
Arguments to **commands** that must be stated explicitly.

Line windows
Parts of line for EDIT to execute subsequent **commands** on.

Memory
This is sometimes known as store and is where a computer stores its data and instructions.

Multi-processing
The execution of two or more **processes** in parallel, that is, at the same tim e.

Output queue
Buffer in memory holding data before being written out to **file.**

Priority
The relative importance of a **process.**

Process
A job requested by the operating system or the user.

Qualifiers
Characters that specify additional conditions for the context in string

Qualified string
A string preceded by one or more qualifiers.

Queue
- see **Output queue.**

Root directory
The top level in the filing system. **Files** and **directories** within the root directory have their names preceded by a colon (:).

Sequential files
A **file** that can be accessed at any point by starting at the beginning and scanning sequentially until the point is reached.

Source file
File being read from.

Syntax
The format or 'grammar' you use for giving a command.

Terminal handler
A **process** handling input and output from the terminal or console.

Volume name
The unique name associated with a disk.

Wild card
Symbols used to match any pattern.

Commodore Business Machines, Inc.
1200 Wilson Drive, West Chester, PA 19380

Commodore Business Machines, Limited
3370 Pharmacy Avenue, Agincourt, Ontario, M1W 2K4