OPEN DESKTOP™

Open Desktop™

Connectivity and DOS Compatibility

OPEN DESKTOP™

The Complete Graphical Operating System

MS-DOS®

Version 3.3

User's Reference

The Santa Cruz Operation, Inc.

Document Number: 12-18-89/1.2.0B
Part Number: 525-210-040
Processed: 4/27/90

Contents

Chapter 1

Introduction

About This Manual

1

This is a reference manual for the MS-DOS® operating system.

The following table is an overview of the topics covered in this manual.

Turn to	If you need to know
Chapter 2	About multi-level directories About paths
Chapter 3	About MS-DOS commands
Chapter 4	How to make a batch file How to use batch commands
Chapter 5	About MS-DOS editing keys
Chapter 6	How to use **edlin**, the line editor
Chapter 7	How to use **edlin** commands
Chapter 8	How to create executable files with **link**
Chapter 9	How to debug files with **debug**
Appendix A	How to configure your system
Appendix B	About ANSI escape sequences About installing device drivers
Appendix C	What a disk error means
Appendix D	What an MS-DOS message means
Appendix E	How to use particular MS-DOS commands

Now that you have seen a brief summary of the topics covered in this manual, you are ready to start with Chapter 2, "About Files and Directories." There you'll learn about some of the more advanced features of MS-DOS.

Chapter 2

About Files and Directories

Chapter 2

About Files and Directories

Introduction

In this chapter you will learn about:

- Protecting and keeping track of your files
- Multi-level directories
- Using wildcards

Protecting Your Files

The MS-DOS operating system is a powerful and useful tool for processing personal and business information. As with any computer, errors may occur and information may be misused. So, if you are doing work that cannot be replaced or that requires a lot of security, you should protect your programs.

You can take simple but effective measures like putting your disks away when you're not using them, or covering the write-protect notch on your program disks.

If your disks contain valuable information, you should make backup copies of them on a regular basis. Another way to protect your programs is by installing your equipment in a secure office or work area.

How MS-DOS Keeps Track of Your Files

MS-DOS stores files in directories. In addition to directories, it uses an area on a disk called the File Allocation Table. When you format a disk with the **format** command, MS-DOS copies this table onto the disk and creates an empty directory, called the root directory. So, on each of your disks, the directories store the files, and the File Allocation Table keeps track of their locations. The table also allocates the free space on your disks so that you have enough room to create new files.

These two system areas, the directories and the File Allocation Table, enable MS-DOS to recognize and organize the files on your disks. To check these areas on a disk for consistency and errors, use the MS-DOS **chkdsk** command.

For example, to check the disk in drive A: type **chkdsk a:**.

In response, MS-DOS displays a status report and any errors it has found, such as files that show a non-zero size in the directory but that really have no data in them.

For an example of such a display and for more information on **chkdsk**, see Chapter 3, "About Commands."

Directories

When there is more than one user on your computer, or when you are working on several different projects, the number of files in the directory can become large and unwieldy. You may want to keep your files separate from a co-worker's, or you may want to organize your programs into convenient categories.

In an office, you can separate and organize files that belong to different people or that relate to specific projects by putting them in different filing cabinets. For example, you might put your accounting programs in one cabinet and your letters in another. You can do the same thing with MS-DOS by putting your files into different directories.

Using directories is one way of dividing your files into convenient groups. These directories can also contain other directories (referred to as subdirectories). Any one directory can contain a maximum of 255 files and directories. This organized file structure is called a multi-level or hierarchical directory system.

The Root Directory

The first level in a multi-level directory is the *root* directory, which is created automatically when you format a disk and start putting files on it. You can create more directories and subdirectories within the root directory.

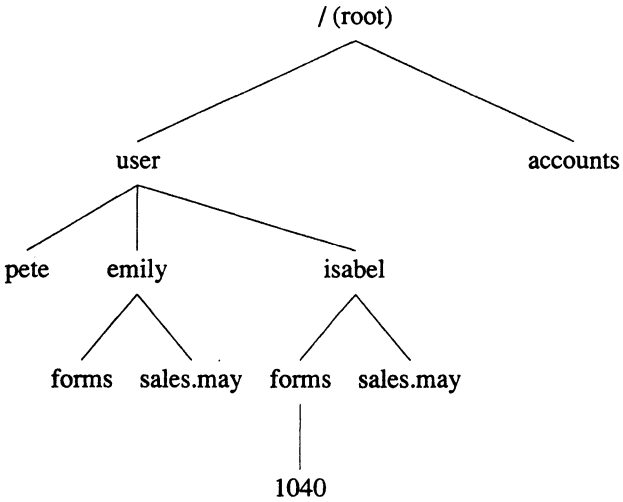
As you create new directories for groups of files, or for other people using the computer, the directory system grows. Within each new directory you can add new files or create new subdirectories.

You can move around in the multi-level system by starting at the root directory and traveling through intermediate subdirectories to find a specific file. Conversely, you can start anywhere within the file system and travel toward the root. Or you can go directly to any directory without traveling through intermediate levels.

Your Working Directory

The directory that you are working in is called the *working* directory. The filenames and commands discussed in this chapter relate to your working directory and do not apply to any other directories in the structure. When you start your computer, you are located in the working directory. Similarly, when you create a file, you create it in the working directory.

Because you can put files in different directories, you and your co-workers can have files with the same names, but with unrelated content. The following figure illustrates a typical multi-level directory structure:



In this example, two subdirectories of the root directory have been created. These subdirectories are:

- A *user* directory containing separate subdirectories for all users of the system.
- A directory containing accounting information, named *accounts*.

As you can see, Pete, Emily, and Isabel each have their own directories, which are subdirectories of the *user* directory. Emily has a subdirectory named *forms*, and both Emily and Isabel have *sales.may* files in their directories, even though Isabel's *sales.may* file is unrelated to Emily's.

This organization of files and directories is not important if you work only with files in your own directory, but if you work with someone else, or on several projects at once, the multi-level directory system becomes handy. For example, you could get a list of the files in Emily's *forms* directory by typing the following command:

dir \user\emily\forms

Note that a backslash (\) separates directories from other directories and files. In the previous example the first backslash includes the root directory. The use of the backslash alone indicates the root directory. For example, the following command displays a list of the files in the root directory:

**dir **

To find out what files Isabel has in her directory, you would type the following command:

dir \user\isabel

This command tells MS-DOS to travel from the root directory to the user directory to the *isabel* directory, and to then display all filenames in the *isabel* directory.

Parent Directories

A parent directory is any directory that contains subdirectories. MS-DOS provides special shorthand notations for the working directory and the parent of the working directory, and automatically creates these two entries whenever you create a directory:

\. MS-DOS uses the shorthand name "." to indicate the name of the working directory in all multi-level directory listings.

\.. These two dots are the shorthand name ".." for the working directory's parent directory (one level up).

If you type the **dir** command followed by two dots, MS-DOS lists the files in the parent directory of your working directory.

If you type the following command MS-DOS lists the files in the parent's parent directory:

dir ..\..

Using Directories

The following sections describe how to display, change, and delete any directory. You will also learn how to create directories and subdirectories.

Creating a New Directory

To create a subdirectory in your working directory, use the **mkdir** (make directory) command. For example, to create a new directory named *user* under your working directory, simply type the following command:

```
mkdir user
```

After MS-DOS runs this command, a new directory will exist under your working directory. You can also make directories anywhere in the directory structure by specifying **mkdir** followed by a path. MS-DOS automatically creates the "." and ".." entries in the new directory.

To put files in the new directory, you can use the MS-DOS line editor, **edlin**. Chapter 6, "edlin: A Line Editor," describes how to use **edlin** to create and save files. You can also create and save files if you have a word processing program such as Microsoft Word®.

Changing Directories

With MS-DOS it is easy to change from your working directory to a different directory, simply type the **chdir** (change directory) command followed by a path. For example, if you type **chdir \user** and then press the <Return> key, MS-DOS changes the working directory to *\user*. You can also specify any path after the command so that you can move around the directory structure. The following command, for example, puts you in the parent directory of your working directory:

```
chdir ..
```

Displaying Your Working Directory

All commands are executed while you are in your working directory. You can find out the name of the directory you are in by typing the MS-DOS **chdir** command with no path. For example, if your working directory is *\user\pete*, when you type **chdir** and press the <Return> key, you would see the following:

```
A:\USER\PETE
```

This is your working drive, A: plus the working directory, *\user\pete*.

You can also type the letters **cd** for the **chdir** command to save time. For example, the following commands are the same:

cd \user\pete
chdir \user\pete

If you want to see the contents of the *\user\pete* directory you can use the MS-DOS **dir** command. The subdirectory might look like this:

```
Volume in drive A has no label
Directory of A:\USER\PETE

.       <Dir>          08-09-86   10:09a
..      <Dir>          08-09-86   10:09a
TEXT    <Dir>          08-09-86   10:09a
FILE1    TXT    5243   08-04-86   09:30a

 4 File(s) 836320 bytes free
```

Note that MS-DOS lists both files and directories in this output. As you can see from the display, Pete has a subdirectory named *text*; the "." refers to the working directory *\user\pete*; the ".." is short for the parent directory *\user*, and *file1.txt* is a file in the *\user\pete* directory. All these directories and files are on the disk in drive A:.

Note

Because files and directories are listed together, you cannot give a subdirectory the same name as a file in that directory. For instance, if you already have a path *\user\pete*, where *pete* is a subdirectory, you cannot create a file named *pete* in the *\user* directory.

Removing a Directory

If you create a directory and decide later that you don't want it any more, you can delete it with the MS-DOS **rmdir** (remove directory) command.

The **rmdir** command lets you delete any directory by specifying its path, but the directory must be empty except for the "." and ".." entries. This prevents you from accidentally deleting files and directories.

To remove all the files in a directory (except for the "." and ".." entries), type **del** followed by the path of the directory. For example, to delete all files in the *\user\emily* directory, type the following command:

del \user\emily

MS-DOS prompts you with the following message:

```
Are you sure (Y/N)?
```

If you really want to delete all the files in the directory, type **Y** (for Yes). If not, type **N** (for No) to stop the command.

Now you can use the **rmdir** command to delete the *\user\emily* directory by typing the following command:

rmdir \user\emily

To save time you can also use the letters **rd** for the **rmdir** command.

Renaming a Directory

There is no command to rename a directory in MS-DOS. You can, however, rename a directory that has no subdirectories. Suppose, for example, you want to rename the *\user\pete* directory and call it *\user\emily*. To do this you would follow these steps (remember to press the <Return> key after each step):

1. To create the new directory, type:

   mkdir \user\emily

2. Then to copy the files from the old directory to the new directory, type:

   copy \user\pete*.* \user\emily

   (This command uses wildcard characters, which are discussed in "Wildcards" later in this chapter.)

3. Now to delete the contents of the old directory, type:

   del \user\pete*.*

   Type **Y** in response to the prompt "Are you sure?"

4. Finally, to remove the old directory, type:

   rmdir \user\pete

Pathnames

When you use multi-level directories, you must tell MS-DOS where the files are located in the directory system. Both Isabel and Emily, for example, have files named *sales.may*, so each would have to tell MS-DOS in which directory her file resides when she wants to use it. This is done by giving MS-DOS a pathname to the file.

A pathname is a sequence of directory names followed by a filename. Each directory name is separated from the previous one by a backslash (\).

The general format of a pathname is as follows:

[*\directoryname*] [*\directoryname...*] *\filename*

A pathname can contain any number of directory names up to a total length of 63 characters. If a pathname begins with a backslash, MS-DOS searches for the file beginning at the root of the directory system. Otherwise, it begins at the working directory and searches along the path from there. Here are two examples:

The pathname of Emily's *sales.may* file is:

\user\emily\sales.may

The pathname of Isabel's *sales.may* file is:

\user\isabel\sales.may

When you are in your working directory, you may interchange a filename with its corresponding pathname. Some sample names are:

\	The root directory.
\accounts	A directory under the root directory that contains accounting files.
\user\isabel\forms\1040	A typical full pathname. This one is for a file named *1040* in the *forms* directory, which belongs to Isabel.
sales.may	A file in the working directory.

Wildcards

If you are using multi-level directories, you will find it easier to search for files on your disks if you use two special characters, called *wildcards*. The wildcard characters are the asterisk (*) and the question mark (?). They are useful in MS-DOS command lines because they give you flexibility when you are specifying paths and files.

A question mark (?) in a filename or filename extension means that any character can occupy that position. The following command, for example, lists all filenames on the default drive that begin with the characters *memo*, that have any character in the next position, that end with the characters *aug*, and that have an extension of *.txt*:

dir memo?aug.txt

Here are some examples of files that might be listed by the above command:

MEMO2AUG.TXT
MEMO9AUG.TXT
MEMOBAUG.TXT

An asterisk (*) used in a filename or filename extension means that any character can occupy that position or any of the remaining positions in the filename or extension. For example, the following command lists all the directory entries on the default drive with filenames that begin with the characters *memo* and that have an extension of *.txt*:

dir memo*.txt

Here are some examples of files that might be listed by this **dir** command:

MEMO2AUG.TXT
MEMO9AUG.TXT
MEMOBAUG.TXT
MEMOJULY.TXT
MEMOJUNE.TXT

Note

The wildcard abbreviation *.* refers to all files in the directory. This feature can be both powerful and destructive when used with MS-DOS commands. For example, the **del** command followed by the wildcard abbreviation *.* deletes all files on the default drive, regardless of filename or extension.

As another example, suppose you want to find a certain accounts file but can't remember its exact name. What you can do is list the directory entries for all files named *accounts* in the default directory of drive A: (regardless of their filename extensions). To do this quickly, you could just type the following command:

dir a:accounts.*

Similarly, to list the directory entries for all files with *.txt* extensions in a directory called *reports* (regardless of their filenames) on the disk in drive B: type the following command:

dir b:\reports*.txt

This command is useful if your text files have *.txt* extensions. For example, by using the **dir** command with wildcard characters, you could get a listing of all your text files – even if you don't remember their filenames. For more information on the **dir** command, refer to Appendix E, "MS-DOS Commands."

Chapter 3

About Commands

Introduction

In this chapter you will learn about:

- Internal and external MS-DOS commands
- Redirecting input and output

There are two types of MS-DOS command:

- Internal commands
- External commands

All of the MS-DOS commands are explained in detail in Appendix E, "MS-DOS Commands," later in this guide.

Internal Commands

Internal commands are the simplest, most commonly used commands, but you cannot see them when you list the directory on your MS-DOS disk because they are part of a file named *command.com*. When you type internal commands, MS-DOS performs them immediately. This is because they were loaded into your computer's memory when you started MS-DOS. Following is a list of the MS-DOS internal commands:

break	**del**	**if**	**ren**	**ver**
chdir	**dir**	**mkdir**	**rmdir**	**verify**
cls	**echo**	**path**	**set**	**vol**
copy	**exit**	**pause**	**shift**	
ctty	**for**	**prompt**	**time**	
date	**goto**	**rem**	**type**	

Using Pathnames with Internal Commands

Some internal commands can use paths and pathnames. Specifically, four commands – **copy**, **dir**, **del**, and **type** – have greater flexibility when you specify a pathname after the command.

The format of the **copy** command is:

copy *pathname pathname*

If the second pathname is a directory, MS-DOS copies all the files you specify in the first pathname into that directory, as in the following example:

copy \user\pete*.* sales

The format of the **del** command is:

del *pathname*

If the pathname is a directory (a path), all the files in that directory are deleted. If you try to delete a path, the prompt "Are you sure (Y/N)?" is displayed. Type **Y** (for Yes) to complete the command, or **N** (for No) to stop the command. For example:

del \user\pete

The format of the **dir** command is:

dir *pathname*

The following command displays all the files and subdirectories for a specific pathname:

dir \user\pete

The format of the **type** command is:

type *pathname*

You must specify a pathname (or filename) for this command. MS-DOS then displays the contents of this file on your monitor in response to the **type** command. For example:

type \user\emily\report.nov

External Commands

Any filename with an extension of *.com*, *.exe*, or *.bat* is considered an external command. For example, files such as *format.com* and *diskcopy.com* are external commands. You can create new command files and add them to MS-DOS. Programs that you create with most languages (including assembly language) will be *.exe* (executable) files. Note, however, that when you use an external command, you do not need to type its filename extension.

Note

If you have more than one external command with the same name, MS-DOS will run only one of them, according to the following order of precedence: *.com*, *.exe*, *.bat*.

Suppose, for example, that your disk includes the files *format.exe* and *format.bat*. If you were to type the external command **format**, MS-DOS would always run the program **format.exe** first. To run the batch file *format.bat*, you would have to place it in a separate directory and give a path along with the external command.

The following is a list of MS-DOS external commands:

append	**device**	**find**	**mode**	**select**	**xcopy**
assign	**diskcomp**	**format**	**more**	**share**	
attrib	**diskcopy**	**graftabl**	**nlsfunc**	**sort**	
backup	**exe2bin**	**graphics**	**print**	**stacks**	
chcp	**fastopen**	**join**	**recover**	**susbst**	
chkdsk	**fc**	**keyb**	**replace**	**sys**	
command	**fdisk**	**label**	**restore**	**tree**	

Note

The **backup** and **restore** commands are used with the C: drive and DOS partitions only. You cannot perform a backup or restore operation to a networked drive.

Using Paths with External Commands

Before MS-DOS can run external commands, it must read them into memory from the disk. When you give an external command, MS-DOS immediately checks your working directory to find that command. If it isn't there, you must tell MS-DOS which directory the external command is in. You do this with the **path** command.

When you are working with more than one directory, you may find it more convenient to put all the MS-DOS external commands in one directory. Then, when it needs them, MS-DOS can quickly find the external commands at one location.

Suppose, for example, that you are in a working directory named *\user\prog* and that the MS-DOS external commands are in *\bin*. To find the **format** command, you must tell MS-DOS to choose the *\bin* path, as in the following command, which tells MS-DOS to search in your working directory and in the *\bin* directory for all commands:

```
path \bin
```

You need only specify this path once during each computer session. Also, if you want to know what the current path is, you can simply type the **path** command by itself. In response, MS-DOS displays the working path on the screen.

You can automatically set your path when you start MS-DOS by including the **path** command in a file called *autoexec.bat*. See Chapter 4, "Batch Processing," for more information on the *autoexec.bat* file.

Redirecting Command Input and Output

Usually, MS-DOS receives input from the keyboard and sends its output to the screen. You can, however, redirect this flow of command input and output. For instance, you may want input to come from a file instead of from the keyboard, and you may want output from a command to go to a file or lineprinter instead of to the screen. With redirection you can also create pipes that let the output from one command become the input for another command.

Redirecting Output

By default, most commands send output to your monitor. If you want to change this and send the output to a file, use a greater-than sign (>) in your command. For example, the following command displays on the screen a directory listing of the disk in the default drive:

dir

The **dir** command can send this output to a file named *contents* if you type the following:

dir > contents

If the *contents* file doesn't exist, MS-DOS creates it and stores your directory listing there. If *contents* does exist, MS-DOS overwrites the file contents with the new data.

Appending Output

If you want to append data to a file rather than overwrite the contents of the file, use two greater-than signs (>>) to tell MS-DOS to append the output of the command (such as a directory listing) to the end of a specified file. For example, the following command appends your directory listing to an existing file named *contents*:

dir >> contents

If *contents* doesn't exist, MS-DOS creates it.

Redirecting Input

Often, it's useful to have input for a command come from a file instead of from the keyboard. This is possible in MS-DOS by using a less-than sign (<) in your command. For example, the following command sorts the file *names*:

sort < names

Redirection of input and output can be combined on a single command line. The following command sorts the file *names* and sends the sorted output to a file called *namelist*:

sort < names > namelist

Chapter 4

Batch Processing

Introduction

In this chapter you will learn how to:

- create a batch file,
- use an *autoexec.bat* file,
- use replaceable parameters in a batch file,
- run a batch file,
- do multitasking with batch files.

4

Note

If you are not interested in writing batch programs, you do not need to read this chapter.

Why Use Batch Files?

You may often find yourself repeatedly typing the same sequence of commands to perform some common task. With MS-DOS you can put this command sequence into a special file called a batch file, and then run the whole sequence of commands by simply typing the name of the batch file. Note that you don't need to type the batch file's extension, even though all your batch files must include the *.bat* extension in their filenames.

MS-DOS performs these "batches" of your commands just as if you had typed them from the keyboard. This is called "batch processing". By using a batch file, you only have to type one command, instead of several. In effect, you use batch files to create personalized commands.

About Batch Processing

Here are a few things you should know before you run a batch process with MS-DOS:

- You must name each batch file with an extension of *.bat*.
- To execute a batch file, you type only its filename and not the extension.
- If you press <CTL>c while the batch file is running, MS-DOS asks you to confirm that you want to terminate the batch process.
- If you remove the disk that contains a batch file being run, MS-DOS prompts you to reinsert the disk so that it can continue processing the file.
- You can specify the name of another batch file as the last command in a batch file. This feature allows you to call another batch file when the first has finished.
- You can use any of the redirection symbols (< > >>) in a batch file. See Chapter 3, "About Commands," for more information on using these symbols.
- You can use the pipe symbol (|) in a batch file, but you cannot pipe in or out of a batch.
- Setting the directory or drive affects every subsequent command in the batch file.
- Setting environment strings also affects every subsequent command in the batch file.

Note

If you have more than one external command with the same name, MS-DOS will run only one of them, according to the following order of precedence: *.com*, *.exe*, *.bat*.

Suppose, for example, that your disk includes the files *format.exe* and *format.bat*. If you were to type the external command **format**, MS-DOS would always run the program *format.exe* first. In order to run the batch file *format.bat*, you would have to place it in a separate directory and give a path along with the external command.

Creating a Batch File

You can create a batch file by using **edlin**, the MS-DOS line editor, or by using the **copy** command. If you want to create files with **edlin**, you should refer to the chapters on **edlin** for more information. The examples in this chapter show you how to use the **copy** command to create batch files.

Suppose, for example, that you want to create a batch file to format and check a new disk. To do this you simply follow these steps:

1. First, type the following:

   copy con checknew.bat

   Press <Return>. This command tells MS-DOS to copy the information from the console (keyboard) to the file *checknew.bat.*

2. Next, type the following lines, pressing <Return> after each:

   rem This is a file to format and
   rem check new disks.
   rem It is named CHECKNEW.BAT.
   pause Insert new disk in drive B:
   format b: /v
   chkdsk b:

3. After the last line, press <CTL>z and then press <Return> to save the batch file. MS-DOS displays the following message:

```
1 File(s) copied
```

   This indicates that the batch file has been created.

Running a Batch File

To execute *checknew.bat*, simply type:

checknew

The result is the same as if the lines in the *.bat* file were entered from the keyboard as individual commands.

The autoexec.bat File

An *autoexec.bat* file lets you run programs automatically when you start MS-DOS. This can be useful when you want to run a specific application under MS-DOS, and when you want MS-DOS to execute a batch program each time you start your computer. By using an *autoexec.bat* file you can avoid loading two separate disks just to perform these tasks.

When you start your computer, MS-DOS searches the root directory of the default disk drive for a file named *autoexec.bat*. If it finds the *autoexec.bat* file, MS-DOS immediately processes it, bypassing the date and time prompts. If MS-DOS does not find an *autoexec.bat* file, then the date and time prompts appear automatically.

Note

MS-DOS does not prompt you for a current date and time unless you include the **date** and **time** commands in your *autoexec.bat* file.

It's a good idea to add these two commands to your *autoexec.bat* file since MS-DOS uses this information to keep your directory current. See Appendix E, "MS-DOS Commands," for more information on the **date** and **time** commands.

Creating an autoexec.bat File

There are many things you can do with an *autoexec.bat* file to help you use MS-DOS more efficiently. For instance, you will probably want to set the time and date, your path, and any other options that you plan to use on a regular basis.

If, for example, you want to automatically load GW-BASIC and run a program called **menu** each time you start MS-DOS, you could create an *autoexec.bat* file as follows:

1. Type the following command and then press <Return>:

   copy con autoexec.bat

   This command tells MS-DOS to copy what you type from the keyboard into the *autoexec.bat* file. Note that you must put the *autoexec.bat* file in the root directory of your MS-DOS disk.

2. Now type the following lines:

   ```
   date
   time
   path=c:\;c:\bin;a:\
   prompt [$p]
   cls
   gwbasic menu
   ```

3. After the last line, press <CTL>**z** and then <Return> to copy these lines into the *autoexec.bat* file.

Once your *autoexec.bat* file is set up as above, it will perform the following actions when you start MS-DOS:

- ask you to enter the date and time,
- set your command search path,
- set your prompt to display the default drive and directory.

Finally, the *autoexec.bat* file will clear the screen and tell MS-DOS to load GW-BASIC and run the **menu** program. To run your own GW-BASIC program, type its name in place of **menu** in the example. In addition to GW-BASIC programs, you can also put any MS-DOS command or series of commands in the *autoexec.bat* file.

Creating a Batch File with Replaceable Parameters

There may be times when you want to create a program and run it with different sets of data. These data may be stored in various MS-DOS files.

With MS-DOS you can create a batch file with replaceable parameters, where a parameter is a command option that you define. These parameters, named **%0** through **%9**, hold the places for the values that you supply when you give the batch command.

Replaceable parameters make batch files more flexible and easy to use. For example, you can create a batch file called *sorter.bat* that sorts a file containing a specific sequence of characters or strings. Each time you execute the *sorter* batch file, you tell MS-DOS which string you want, which file to search to find that string, and which temporary file to use for sorting. The *sorter* file would then print the resulting list on the printer.

1. To create the *sorter.bat* file, enter the following command:

   copy con sorter.bat

2. Now type the following lines:

   ```
   type %2 | find "%1" > %3
   type %3 | sort > prn
   del %3
   ```

3. To save the batch file, press <CTL>z and then <Return>.

The batch file *sorter.bat* now consists of three command lines and is on the disk in the default drive.

When you execute the file, MS-DOS sequentially replaces **%1**, **%2**, and **%3** with the parameters you supply. If you use the parameter **%0**, MS-DOS always replaces it with the drive name (if specified) and the filename of the batch file (for example, *sorter*).

Creating a Batch File with Replaceable Parameters

Keep the following in mind when creating batch files with replaceable parameters:

- You can specify up to ten replaceable parameters (**%0** through **%9**). If you want to specify more than ten, refer to the **shift** command later in this chapter.

- If you use the percent sign as part of a filename within a batch file, you must type it twice. For example, to specify the file *abc%.exe*, you must type it as *abc%%.exe* in the batch file.

Running a Batch File with Replaceable Parameters

To run the batch file *sorter.bat*, type the batch filename followed by the parameters that you want MS-DOS to substitute for **%1**, **%2**, and **%3**.

Suppose that on the disk in drive A: you have a file that lists your customers' names and regions. The file might look something like this:

```
Shores, Sandy      north

Poster, Emily      south

Sharpe, Isabel R.  north

Fisher, Pete       east

Conrad, Betty      south

Rey, Fernando      north

Shaw, Rick         west

Moss, Chris        north
```


If you want to print an alphabetical list of the customers in the north, you can run the *sorter* batch file, with the appropriate parameters, by entering the following command:

sorter north a: customer temp.fil

The output on the printer should look like this:

Moss, Chris north
Rey, Fernando north
Sharpe, Isabel R. north
Shores, Sandy north

The following table shows how MS-DOS replaces each of the parameters in the previous example:

Batch filename (%0) *sorter*
Parameter1 (%1) *north*
Parameter2 (%2) *a:customer*
Parameter3 (%3) *temp.fil*

The result is the same as if you had typed each of the commands in *sorter* with its parameters, as follows:

```
type a:customer | find "north" > temp.fil
type temp.fil | sort > prn
del temp.fil
```

Using the batch file, however, saves typing time and is much easier to remember.

Using Temporary Files

When using batch files, you may often want to use a temporary file to hold your work. You could use the same name each time you wanted to use a temporary file.

However, if you are using more than one batch file that uses the same temporary file, you might lose the contents of this temporary file. To avoid this problem, you should use a replaceable parameter to specify the name of the temporary file. Then each time you run the batch file, you'll be able to substitute a unique filename and you won't have to worry about information from one batch file getting into another.

It's also a good idea to delete temporary files once you have finished using them. Otherwise, these files would eventually take up all the space on your disk.

Batch Processing Commands

Now you have seen some of the capabilities of batch files. In this section you'll find out how to add power and flexibility to your batch programs by using batch processing commands. The following table lists these batch commands and describes briefly what they do:

Command	Function
echo	Turns the batch file echo feature on or off, or displays the current setting.
for	Performs a command for a set of files.
goto *label*	Processes commands starting with the line after the specified *label*.
if	Performs a command if a condition is met.
pause	Pauses during the processing of a batch file.
rem	Displays a comment in a batch file.
shift	Increases the number of replaceable parameters in a batch process.

Each of these commands is described in the pages that follow.

echo

The **echo** command turns the batch echo feature on and off.

Syntax

echo [ON]

or

echo [OFF]

or

echo [message]

Description

Normally, commands in a batch file are displayed ("echoed") on the screen when they are received by MS-DOS. You can turn off this feature by using the OFF option with the **echo** command. Similarly, you can turn the **echo** feature back on by using the ON option with **echo**.

If you do not specify ON or OFF, **echo** displays the current setting.

The command **echo** *message* (where *message* is a line of text) is only useful if **echo** is off and if you are using a batch file. If, in your batch file, you type the **echo** command followed by a message, you can print messages on your screen. You can also put several **echo** *message* commands in your batch file to display a message that is several lines in length.

Example

The following is an example of a batch file message of more than one line:

```
echo off
echo This batch file
echo formats and checks
echo new disks.
```

for

The **for** command performs a command for a set of files.

Syntax

for %%c in *set* do *command*
(for batch processing)

for %c in *set* do *command*
(for interactive processing)

Description

To avoid confusion with the **%0** through **%9** batch parameters, the variable **c** can be any character except 0,1,2,3,...,9.

set is (**item***)

The **set is** command sequentially sets the **%%c** variable to each member of *set*, and uses the variable to evaluate *command*. If a member of *set* is an expression involving a wildcard (* or ?), then the variable is set to each matching item from the disk. In this case, only one such item is in *set*, so the command ignores any item other than the first.

Examples

The following example binds the variable **%f** to files ending with **.asm* in the working directory.

for %%f in (**.asm*) **do masm %%f**

It then executes a command of the following form:

masm *filename*

filename could be any one of the following:

invoice.asm
receipts.asm
taxes.asm

The following example binds the variable **%f** to the files named *report*, *memo*, and *address*; it then deletes each of these files:

for %%f in (*report memo address*) **do del %%f**

You must use two percent signs (**%%**) so that one will remain after the batch parameter (**%0** through **%9**) processing is complete. If you had only **%f**, instead of **%%f**, then the batch parameter processor would see the **%**, look at **f**, decide that **%f** was an error (a bad parameter reference), and throw out the **%f** so that the **for** command would never see it.

If the **for** command is not in a batch file, you should use only one percent sign.

goto

The **goto** *label* command processes commands starting with the line after the specified label.

Syntax

goto *label*

Description

goto lets you take commands from the batch file beginning with the line after the *label*, where a label is defined as the characters following **goto**. This label may include spaces, but not other separators, such as semicolons or equal signs. If your batch file does not contain the label, the batch file terminates.

Note

Any line in a batch file that starts with a colon (:) is ignored during batch processing.

Example

The following example sends the program processor to the label named *end* only if no errors occur when you format the disk in drive A:

```
:begin
echo off
format a: /s
if errorlevel 0 goto end
echo An error occurred during formatting.
:end
echo End of batch file.
```

if

The **if** command performs a command based on the result of a condition.

Syntax

if [NOT] **errorlevel** *number command*

or

if [NOT] **string1 == string2** *command*

or

if [NOT] **exist** *filename command*

Description

The **if** statement allows conditional execution of commands. When the condition is true, MS-DOS executes the command, otherwise it ignores the command.

The conditions are described as follows:

errorlevel *number*

True if, and only if, the previous program executed by *command.com* had an exit code of *number* or higher. (When a program finishes, it returns an exit code via MS-DOS.) You can use this condition to perform other tasks that are based on the previous program's exit code.

string1 == string2

True if, and only if, **string1** and **string2** are identical after parameter substitution. Strings may not contain separators, such as commas, semicolons, equal signs, or spaces.

exist *filename*

True if, and only if, *filename* exists.

If you specify the NOT parameter, MS-DOS executes the command when the condition is false.

Example

The following example prints the message "can't find datafile" if the file *product.dat* does not exist on the disk:

```
if not exist product.dat echo can't find datafile
```

pause

The **pause** command suspends execution of a batch file.

Syntax

pause [*comment*]

Description

When a batch file is running, you may need to change disks or perform some other action. The **pause** command suspends execution of the batch file until you press any key, unless you press the <CTL>**c** key sequence.

When the command processor encounters **pause**, it prints the following message:

```
Strike a key when ready . . .
```

If you press <CTL>**c**, MS-DOS displays the following message:

```
Terminate batch job (Y/N)?
```

If you type **Y** in response to this prompt, the batch file ends and control returns to the operating system. Therefore, you can use **pause** to divide a batch file into pieces that allow you to end the batch command file at any intermediate point.

The comment parameter is useful when you want to display a special message. Unless **echo** is off, **pause** displays this comment before the "Strike a key" message.

Note

The **pause** and **comment** line of your batch file will not appear if **echo** is off.

Example

Suppose you want a program to display a message that asks the user to change disks in one of the drives. To do this you might use the following command:

pause Please put a new disk into drive A

If **echo** is on, this line will precede the "Strike a key" message when you run the batch file.

rem

During execution of a batch file, **rem** displays remarks that are on the same line as the **rem** command in that batch file.

Syntax

rem [*comment*]

Description

The comment parameter is a line of text that helps you identify and remember what your batch file does.

The only separators allowed in the comment are spaces, tabs, and commas.

In your batch file, you can use **rem** without a comment to add spacing for readability.

Example

The following example shows a batch file that uses remarks for both explanation and spacing:

```
rem This file formats and checks new disks
rem It is named checknew.bat
rem
pause Insert new disk in drive B
format B: /v
chkdsk B:
```

shift

The **shift** command lets you change the position of replaceable parameters in batch file processing.

Syntax

shift

Description

You can use the **shift** command to change the positions of (replaceable) command line parameters.

Usually command files are limited to handling ten parameters, **%0** through **%9**. By using **shift**, you can access more than ten parameters. This means that if there are more than ten parameters given on a command line, those that appear after the tenth (**%9**) will be shifted one at a time into **%9**.

You can use the **shift** command even if you have less than ten parameters.

There is no backward **shift** command. Once you have executed **shift**, you cannot recover the first parameter (**%0**) that existed before the **shift** command.

Example

The following file *mycopy.bat* shows how to use the **shift** command with any number of parameters. It copies a list of files to a specific directory.

```
rem mycopy.bat copies
rem any number of files
rem to a directory.
rem The command is
rem mycopy dir files
:one
if ''%1'' = '' '' goto two
set todir = %1
shift
copy %1 %todir%
goto one
:two
set todir=
echo All done
```

Chapter 5

MS-DOS Editing Keys

Introduction

In this chapter you will learn about:

- the MS-DOS editing and function keys,
- the editing template,
- the MS-DOS control characters.

Many operating systems handle command input differently to MS-DOS. One difference in particular that sets MS-DOS apart is its set of special editing keys. For instance, with MS-DOS you don't have to type the same sequences of keys repeatedly, because the most recently typed command line is automatically placed in a special storage area called a template.

By using the template and the special editing keys, you can take advantage of the following MS-DOS features:

- You can repeat a command instantly by pressing two keys.
- If you make a mistake in a command line, you can edit and retry it without having to retype the entire line.
- With a minimum of typing, you can edit and execute a command line that is similar to a previous one.

When you type a command and press the <Return> key, MS-DOS automatically sends it to the command processor (*command.com*) for execution. At the same time, MS-DOS also sends a copy of this command to the template. You can then recall or modify the command by using the MS-DOS special editing keys.

The MS-DOS Editing Keys

Key	Editing function
<F1>	Copies one character from the template to the command line.
<F2>	Copies characters up to the character specified in the template and puts these characters on the command line.
<F3>	Copies all remaining characters in the template to the command line.
<DEL>	Skips over (does not copy) a character in the template.
<F4>	Skips over (does not copy) the characters in the template up to the character specified.
<ESC>	Voids the current input and leaves the template unchanged.
<Ins>	Enters/exits insert mode.
<F5>	Makes the new line the new template.
<F6>	Puts a <CTL>z (1AH) end-of-file character in the new template.

Using the Template

Suppose you want to see the directory information for a file named *invest.mnt*. To get this information you could type the following command:

dir invest.mnt

This command line (**dir invest.mnt**) is also saved in the template. If you want to repeat the command, just press two keys: <F3> and <Return>.

MS-DOS displays the repeated command on the screen when you press <F3> as shown below:

```
dir invest.mnt
```

When you press the <Return> key, the command line is sent to the command processor for execution.

If you want to display information about a file named *invest.rpt*, you can use the contents of the template. Pressing <F2> followed by the letter *m* copies all characters from the template to the command line, up to but not including the *m*. MS-DOS displays:

```
dir invest._
```


Note that the underline is your cursor. Now type the letters **rpt** to get the following result:

```
dir invest.rpt_
```

The command line (**dir invest.rpt**) is now in the template and ready to be sent to the command processor for execution. To run the command, press the <Return> key.

Now, assume that you want to run the following command:

type invest.rpt

To do this, type the word **type** and then press the following sequence of keys: <Ins>, <Space>, <F3>, <Return>.

As you type, the characters appear directly on the command line, overwriting their corresponding characters in the template. Before you press the <Ins> key, the word "type" replaces the word "dir" (and the

space following it) in the template. After you press the <Ins> key, this automatic replacement feature is turned off.

To insert a space between the word "type" and the filename *invest.rpt*, you pressed <Ins> and then <Space>. Finally, to copy the rest of the template to the command line, you pressed <F3> and then the <Return> key.

The command line **type invest.rpt** has been processed by MS-DOS, and the template now looks like this:

```
type invest.rpt
```

Correcting Errors in the Template

If you had misspelled "type" as "pyte", for example, a command error would have occurred. Still, instead of throwing away the whole command, you could save the misspelled line before pressing the <Return> key. You can do this by pressing the <F5> key before the <Return> key, creating a new template:

```
pyte invest.rpt
```

You can then edit this error by pressing the following keys:

<DEL> <DEL> <F1> <Ins> yp <F3>

To illustrate how the keys you type affect the command line, compare the key pressed with its result (shown alongside it, together with a description of the effect).

DEL —
Skips over 1st template character

DEL —
Skips over 2nd template character

<F1> t_
Copies 3rd template character

<Ins> yp
typ_ Inserts two characters, y and p

<F3> type invest.rpt_
Copies rest of template

Notice that <DEL> does not affect the command line. Instead, it affects the template by deleting the first character. Similarly, <F4> deletes characters in the template, up to, but not including, a given character.

These special editing keys give you more power and flexibility when you are typing. But, in addition to these keys, MS-DOS also has control characters that help you control the output from a command, or control the contents of the current command line. The next section describes how to use the MS-DOS control characters.

Using the MS-DOS Control Characters

A control character affects the command line in a special way. For example, you use <CTL>**c** to stop running the current command, and you use <CTL>**s** to suspend the screen output from a command.

Note

When you type a control sequence, such as <CTL>**c**, you must hold down the <CTL> key and then press the **c** key.

The following table shows the MS-DOS control characters and describes what they do.

Keys	Action
<CTL>**c**	Aborts the current command.
<CTL>**h**	Removes the last character from a command line, and erases that character from the terminal screen.
<CTL>**j**	Inserts a physical end-of-line, but does not empty the command line. Use the <Linefeed> key to extend the current logical line beyond the physical limits of the terminal screen.
<CTL>**n**	Causes echoing of output to a lineprinter.
<CTL>**p**	Causes terminal output to a lineprinter.
<CTL>**s**	Suspends output display on the screen. Press <CTL>**s** again to resume.
<CTL>**x**	Cancels the current line, empties the command line, and then outputs a backslash(\), <Return>, and <Linefeed>. <CTL>**x** does not affect the template used by the special editing commands.

Chapter 6

edlin: A Line Editor

Introduction

In this chapter you will learn:

- how to start **edlin**, the line editor program;
- how to quit **edlin**;
- how to use the MS-DOS special editing keys with **edlin**.

For information on specific **edlin** commands, see Chapter 7, "edlin Commands."

edlin Basics

You can use the MS-DOS line editor, **edlin**, to create text files and save them on your disks, or to update existing files, saving both the original and the updated files. Also, with **edlin** you can delete, edit, insert, and display lines in files. It will also help you search for, and delete or replace, text within your files.

Though it isn't a word processor, **edlin** does make it easy for you to create and revise files such as memos, letters, reports, or GW-BASIC programs.

edlin divides the text from a file into lines, each line containing up to 253 characters. It gives each line a number and always numbers the lines consecutively. Even though you see these line numbers on the screen when you use **edlin**, they are not part of the file.

When you insert lines of text in a file, the line numbers are adjusted automatically. Similarly, when you delete lines in a file, the lines are renumbered automatically.

Starting edlin

To start **edlin**, you simply type **edlin** *filename*. If you are creating a new file, "filename" should be the name or pathname of the file you wish to create. If **edlin** does not find this file on the default disk drive, it creates a new file with the name or pathname that you specify. For example, if you want to create a file called *budget.jun*, you would type the following command and then press the <Return> key:

edlin budget.jun

edlin would then display the following:

```
New file
*_
```

Note that the **edlin** prompt is an asterisk (*).

To begin entering text you must type an **I** (insert) to insert lines. The **I** command is discussed in Chapter 7, "edlin Commands". For now you can type lines of text into your file, or use any of the **edlin** commands. These are discussed in more detail in Chapter 7, "edlin Commands."

6

Note

Be sure to press the <Return> key at the end of each line.

Suppose you want to edit an existing file called *budget.may*. To do this you would type the following:

edlin budget.may

Then, when **edlin** finds the *budget.may* file, it loads it into memory. If your computer has enough memory to load the entire file, **edlin** displays the following message:

```
End of input file
*
```

You can then edit the file by using **edlin** commands.

If the file is too large to be loaded into memory, **edlin** loads lines from the file until memory is 3/4 full, and displays the asterisk (*) prompt. You can then edit the portion of the file that is in memory.

To edit the rest of the file, you must save some of the edited lines on a disk to free memory. **edlin** will then be able to load the remaining unedited lines from the disk into memory. Refer to the **W** (write) and **A** (append) commands in Chapter 7, "edlin Commands," for instructions on editing large files.

Quitting edlin

When you finish your editing session, you can save your original file and the updated (new) file by using the **E** (end) command, discussed in Chapter 7, "edlin Commands." **edlin** renames your original file with the extension *.bak*, and saves the updated file with the filename and extension you gave when you started **edlin**.

Note

You cannot update a file with an extension of *.bak* because when you try to save your file, **edlin** will always save the original file as *.bak*, thereby losing your changes. If you need to edit such a file, rename it with another extension (using the MS-DOS **ren** command, see Appendix E "MS-DOS Commands"), and start **edlin** by using the new filename.

Special Editing Keys

To edit your text files you can also use the special editing keys and template introduced in Chapter 5, "MS-DOS Editing Keys."

The following table summarizes the commands, codes, and functions of the special editing keys. Descriptions of these special editing keys follow the table:

Key	What it does
<F1>	Copies one character from the template to the new line.
<F2>	Copies all characters from the template to the new line, up to the character specified.
<F3>	Copies all remaining characters in the template to the screen.
<DEL>	Does not copy (skips over) a character.
<F4>	Does not copy (skips over) the characters in the template, up to the character specified.
<ESC>	Clears the current input and leaves the template unchanged.
<Ins>	Enters/exits insert mode.
<F5>	Makes the new line the new template.

<F1>

When you press the <F1> key, **edlin** copies one character from the template to the current line, and turns insert mode off.

As an example of how to use the <F1> key with **edlin**, type the following line:

```
1:  *Sharpe Office Supplies.
2:  *_
```

At the beginning of the editing session, the cursor (shown by the underline) is at the beginning of the line. When you press the <F1> key, **edlin** copies the first character, *S*, to line 2 as shown here:

```
2:  *S_
```

Each time you press the <F1> key one more character appears. First <F1>:

```
2:  *Sh_
```

Second <F1>:

```
2:  *Sha_
```

Third <F1>:

```
2:  *Shar_
```

<F2>

When you press the <F2> key, **edlin** copies all the characters, up to a given character, from the template to the current line. The given character is the one that you type immediately after <F2>. **edlin** does not copy or display the given character on the screen, but it does copy and display the characters from the template up to the position of that character. Of course, if the template does not contain the character, **edlin** does not copy anything.

If you use the <F2> key, you automatically turn off insert mode.

As an example of how to use the <F2> key with **edlin**, type the following line:

```
1:  *Sharpe Office Supplies.
2:  *_
```

Remember that at the beginning of the editing session, the cursor (shown by the underline) is at the beginning of the line. When you press the <F2> key followed by the letter **c**, **edlin** copies the characters up to the *c* in the word *Office*, as shown below:

```
2:  *Sharpe Offi_
```

<F3>

When you press the <F3> key, **edlin** copies the remaining characters in the template to the current line. No matter where the cursor is when you press the <F3> key, **edlin** displays the rest of the line and leaves the cursor at the end of the line.

As an example of how to use the <F3> key with **edlin**, type the following line:

```
1:  *Sharpe Office Supplies.
2:  *_
```

Remember that at the beginning of the editing session, the cursor (shown by the underline) is at the beginning of the line. When you press the <F3> key, **edlin** copies the characters in the template (shown in line 1) to the line with the cursor (shown in line 2):

```
2:  *Sharpe Office Supplies._
```

This command automatically turns off insert mode.

<DEL>

Each time you press the <DEL> key, **edlin** skips over and does not copy the next character in the template. The action of the <DEL> key is similar to the <F1> key, except that <DEL> skips a character in the template instead of copying it to the current line.

As an example of how to use the <DEL> key with **edlin**, type the following line:

```
1:  *Sharpe Office Supplies.
2:  *_
```

Remember that at the beginning of the editing session, the cursor (shown by the underline) is at the beginning of the line. When you press the <DEL> key, **edlin** skips over the first character, *S*. The cursor does not move as **edlin** changes the template. To see how much of the line has been skipped over, press the <F3> key. This action moves the cursor past the last character of the line, and you see:

```
2:  *harpe Office Supplies._
```

<F4>

When you press the <F4> key, **edlin** skips over all characters up to a given character in the template. **edlin** does not copy or display any of the characters up to and including the given character. Of course, if the template does not contain that character, **edlin** does not skip any characters.

Note that the action of the <F4> key is similar to that of the <F2> key, except <F4> skips over characters in the template instead of copying them to the current line.

As an example of how to use the <F4> key with **edlin**, type the following line:

```
1:  *Sharpe Office Supplies.
2:  *_
```

Remember that at the beginning of the editing session, the cursor (shown by the underline) is at the beginning of the line. When you press the <F4> key followed by the letter **c**, **edlin** skips over all the characters in the template up to the *c* in the word *Office*. The cursor does not move as **edlin** changes the template. To see how much of the line has been skipped over, press the <F3> key to copy the template. This action displays the rest of the line and moves the cursor to the end of the line:

```
2:  *ce Supplies._
```

<ESC>

The <ESC> key quits input and clears the current line. When you press the <ESC> key, **edlin** empties the current line and leaves the template unchanged. <ESC> also prints a backslash (\), <Return>, and <Linefeed>, and turns insert mode off. The cursor (shown by the underline) is at the beginning of the line. If you then press the <F3> key, **edlin** copies the template to the current line, making it appear as it was before you pressed the <ESC> key.

As an example of how to use the <ESC> key with **edlin**, type the following lines:

```
1: Sharpe Office Supplies.
2: *The World Leader_
```

To cancel the current line, line 2, press <ESC>. Notice that a backslash appears on line 2 to tell you it has been canceled:

```
2: *The World Leader\
```

Press the <Return> key to keep line 1, or to perform any other editing functions. Now if you press <F3>, **edlin** copies the original template to the line:

```
2: Sharpe Office Supplies.
```

<Ins>

The <Ins> key enters insert mode or replace mode. When you start **edlin**, you are automatically in replace mode. The first time you press the <Ins> key, **edlin** enters insert mode. In insert mode the cursor in the template does not move, but in the current line it moves as you insert each character. When you finish inserting characters and press the <Ins> key again, **edlin** re-enters replace mode with the cursor at the same character in the template as when you entered insert mode.

In insert mode, **edlin** puts characters that you type at the keyboard into the template and in front of the cursor on the current line.

In replace mode, **edlin** replaces characters in the template and on the current line with characters that you type at the keyboard.

As an example of how to use the <Ins> key with **edlin**, type the following line:

```
1:  *Sharpe Office Supplies.
2:  *_
```

Remember that at the beginning of the editing session, the cursor (shown by the underline) is at the beginning of the line. First, press the <F2> key followed by the letter **O**. Line 2 becomes:

```
2:  *Sharpe _
```

Second, press the <Ins> key and type **Automatic** followed by a space. The word *Automatic* is added to line 2:

```
2:  *Sharpe Automatic _
```

If you now press the <F3> key, **edlin** copies the rest of the template to the line:

```
2:  *Sharpe Automatic Office Supplies._
```

If you press the <Return> key after entering insert mode, **edlin** does not copy the remainder of the template and ends the current line after the inserted text:

```
1:  Sharpe Office Supplies.
2:  *Sharpe Automatic
3:  *_
```

To exit insert mode and enter replace mode, simply press the <Ins> key again. Now all the characters you type will replace characters in the template.

For a further example of how to use the <Ins> key, type the following line:

```
1:  *Sharpe Office Supplies.
2:  *_
```

Once again, remember that at the beginning of the editing session, the cursor (shown by the underline) is at the beginning of the line. Type <F2> followed by the letter **c**, and line 2 becomes:

```
2:  *Sharpe Offi_
```

Now type <Ins> **cial**, to get the following line:

```
2:  *Sharpe Official_
```

Finally, type <Ins> **Sharpeware.**, and you have:

```
2:  *Sharpe Official Sharpeware._
```

Notice that you inserted *cial* and replaced *ce Supplies* with *Sharpeware.* to get *Sharpe Official Sharpeware.*

<F5>

The <F5> key creates a new template. When you press the <F5> key, **edlin** copies the current line to the template and deletes the previous contents. Pressing <F5> also displays an @ ("at" symbol), and outputs a <Return> and a <Linefeed>. When you press <F5>, **edlin** empties the current line and turns off insert mode.

Note

<F5> performs the same function as the <ESC> key, except that it changes the template, printing an @ instead of a backslash.

As an example of how to use the <F5> key with **edlin**, type the following line:

```
1:  *Sharpe Office Supplies.
2:  *_
```

Remember that, at the beginning of the editing session, the cursor (shown by the underline) is at the beginning of the line. Press the <F2> key followed by the letter **c**. Line 2 becomes:

```
2:  *Sharpe Offi_
```

Now type <Ins> **cial**, and you get:

```
2:  *Sharpe Official_
```

Then, typing <Ins> **Sharpeware.** gives you:

```
2:  *Sharpe Official Sharpeware._
```

At this point, suppose you want to add a word at the beginning of this line, but you don't want to backspace and retype the whole line. Just press the <F5> key to put the current line into the template. You see:

```
1:  Sharpe Office Supplies.
2:  *Sharpe Official Sharpeware.@
```

The @ shows that this new line is now the new template. To add the word *Introducing:*, followed by a space, at the beginning of the line, enter <Ins> **Introducing:** . This gives you:

```
2:  *Introducing: _
```

Then press the <F3> key to insert the contents of the template:

```
2:  *Introducing: Sharpe Official Sharpeware._
```

Chapter 7

edlin Commands

Introduction

This chapter describes in detail the **edlin** commands, listed in the following table:

Command	Name	What it does
A	append	Appends lines.
C	copy	Copies lines.
D	delete	Deletes lines.
line no.	edit	Edits a line or lines.
E	end	Ends editing.
I	insert	Inserts lines of text.
L	list	Lists a range of lines.
M	move	Moves a range of text to a specified line.
P	page	Pages through a file 23 lines at a time.
Q	quit	Quits the editing session without saving the file.
R	replace	Replaces text.
S	search	Searches for text.
T	transfer	Transfers the contents of another file into the file being edited.
W	write	Writes specified lines to disk.

Some Tips for Using edlin Commands

Once you have started editing a file with **edlin**, you can use the **edlin** commands to edit lines of text in it. Here are a few things to remember when using **edlin** commands:

- You can use pathnames in commands. For example, by typing the following command you can edit a file named *report.may* in a subdirectory named *\sharpe\budget*:

  edlin \sharpe\budget\report.may

- You can reference line numbers relative to the current line, which **edlin** shows with an asterisk (*). To indicate lines before the current line, use a minus sign with a number; to indicate lines after the current line, use a plus sign with a number. The following command, for example, lists 10 lines before the current line, and 10 lines after the current line:

  -10,+10L

Note

A capital "L" is used here for the **L** (list) command to avoid confusion with the number one. A lowercase "l" would work just as well.

- You can type **edlin** commands with or without a space between the line number and command. For example, for deleting line 6, the command **6d** is the same as **6 d**.

- You can type multiple commands on one command line. Just type them one after another. However, if you want to use the **line** (edit) command to edit a specific line, you have to separate the line number from the other commands with a semicolon.

Note

In the following examples, you will see control key sequences. To enter a control sequence, press the <CTL> key and hold it down while pressing the appropriate character (**z**, **c**, or **v**).

To end a string in an **S** (search) or **R** (replace) command, just press <CTL>**z** instead of the <Return> key. The following command line, for example, edits line 15, then displays lines 10 through 20 on the screen:

15;-5,+5L

The command line in the next example searches for the words *monthly budget* and then displays five lines before and five lines after the line that contains *monthly budget*:

monthly budget<CTL>Z-5,+5L

If **edlin** can find any lines that contain the words *monthly budget*, the displayed lines are the five lines before and the five lines after the current line. Note that the current line (the line with the asterisk) must be before the line or lines that contain the search string.

You can insert a control character, such as <CTL>**c**, into text by using the quote character, <CTL>**v**, before it while in insert mode. <CTL>**v** tells MS-DOS to recognize the next capital letter typed as a control character. You can also put a control character in an **S** (search) or **R** (replace) command by using the quote character. For example, the following command finds the first occurrence of <CTL>**z** in a file:

s<CTL>v z

The next example replaces all occurrences of <CTL>**z** in a file with *pencil*:

r<CTL>vz<CTL>z pencil

And this next command replaces all occurrences of <CTL>**c** with *pen*:

s<CTL>vc<CTL>z pen

It is also possible to insert <CTL>v into the text by typing <CTL>vv.

- The <CTL>z character ordinarily tells **edlin**, "This is the end of the file." If you have <CTL>z characters elsewhere in your file, you must tell **edlin** that these other control characters do not mean end-of-file. To tell **edlin** to ignore the <CTL>z characters in the file and to show you the entire file, use the **/b** switch. For example, when you start **edlin** and want to ignore all <CTL>z characters in a file, you could use the **/b** option with the **edlin** command and your filename.

edlin Command Options

Many **edlin** commands accept one or more options. The effect of a command option varies, depending on which command you use it with. The following list describes each option:

- **The Line Option**

  The line option is a line number that you type. Use a comma or space to separate the numbers from other line numbers, other options, and from the command.

  You can specify a line in one of four ways:

  1. number

     Any number less than 65534. If you specify a number larger than the largest existing line number, then line refers to the line after the last line number.

  2. . (period)

     If you specify a period for line, it refers to the current line number. The current line is the last line you edited, not necessarily the last line you displayed. **edlin** marks the current line with an asterisk (*) between the line number and the first character.

  3. # (cross hatch)

     The cross hatch indicates the line after the last line number. If you type # for line, it is the same as typing the last line number plus one.

  4. <Return> key

     If you type a command and then press the <Return> key without any of the line markers in this list, **edlin** uses a default value for each command (default values may be different for each command).

- **The Question Mark Option**

  The question mark (?) option tells **edlin** to ask you if the correct string has been found. You use the question mark only with the **R**

(replace) and **S** (search) commands. Before continuing, **edlin** waits for you to type a **Y** or press the <Return> key for a "Yes" response, or to press any other key for a "No" response.

- **The Text Option**

  The text option specifies text to be found, to be replaced, or to replace other text. Use the text option only with the **S** (search) and **R** (replace) commands. You must end each string of text with a <CTL>**z** or a <Return> (see the **R** command for details). You should not leave any spaces between strings of text or between a string of text and its command letter, unless you want those spaces to be part of the text.

edlin Commands

The following sections describe the **edlin** commands. Each description explains the purpose and correct usage (syntax) of the command. Also, each command has several comments and examples, which offer advice, help, and even some shortcuts for using **edlin**.

append

The **append** command adds a specified number of lines from your disk to the file being edited in memory.

Syntax

[*n*]**a**

Description

The *n* option specifies the number of lines that you want to read into memory. You need this command only if the file you want to edit is too large to fit into memory. When you start **edlin**, it reads as many lines as possible into memory.

To edit the remainder of the file that will not fit into memory, you must write lines that you have already edited onto your disk. Then you can load unedited lines from your disk into memory by using the **a** (append) command. Refer to the **w** (write) command in this chapter for information on how to write edited lines to your disk.

7

Note

If you do not specify the number of lines to append, **edlin** adds lines to the available memory until it is ¾ full, but does nothing if available memory is already ¾ full.

After the **a** command reads the last line of the file into memory, **edlin** displays the message "End of input file."

copy

The **copy** command copies a range of lines to a specified line number, and when used with the *count* option, copies this range as many times as you want.

Syntax

[*line*],[*line*],*line*[,*count*]**c**

Description

The first and second *line* options specify the range of lines that you want to copy. If you omit the first or second *line* option, **edlin** defaults to the current line. The third *line* option specifies the line before which **edlin** will place the copied lines.

You must not overlap the line numbers or you will get an "Entry error" message. For example, the following command would result in an error message:

3,20,15c

If you do not specify a number for the *count* option, **edlin** copies the lines one time and automatically renumbers the file after the copy.

Examples

Type the following lines:

```
1:  Sharpe Office Supplies
2:  The World Leader in Office Sharpeware
3:  I.R. Sharpe, President
4:
5:  "You oughta be Sharpe too."
```

You could copy this entire block of text by typing the following command:

1,5,6c

This command copies lines 1 through 5 and duplicates them, beginning on line 6. The result is:

```
1:    Sharpe Office Supplies
2:    The World Leader in Office Sharpeware
3:    I.R. Sharpe, President
4:
5:    "You oughta be Sharpe too."
6:    Sharpe Office Supplies
7:    The World Leader in Office Sharpeware
8:    I.R. Sharpe, President
9:
10:   "You oughta be Sharpe too."
```

Now if you want to place this text within other text, you would specify the third *line* option as the line you want the copied text to appear before. For example, suppose you want to copy lines and insert them in the middle of the file. The command **7,10,5c** would result in the following:

```
1:    Sharpe Office Supplies
2:    The World Leader in Office Sharpeware
3:    I.R. Sharpe, President
4:
5:    The World Leader in Office Sharpeware
6:    I.R. Sharpe, President
7:
8:    "You oughta be Sharpe too."
9:    "You oughta be Sharpe too."
10:   Sharpe Office Supplies
11:   The World Leader in Office Sharpeware
12:   I.R. Sharpe, President
13:
14:   "You oughta be Sharpe too."
```

delete

The **delete** command deletes a specified range of lines in a file.

Syntax

[*line*][,*line*]**d**

Description

If you omit the first *line* option, **edlin** defaults to the current line (the line with the asterisk next to the line number). If you omit the second *line* option, then **edlin** deletes just the first line. Remember, too, that when you delete lines, **edlin** automatically renumbers the file.

Examples

Suppose that the following file exists and is ready to edit:

```
1:    Dear Mr. Dimm,
2:
3:    I was sorry to hear of your recent
4:    hospitalization due to electrical
5:    shock from our X-1000 Automatic
6:    Pencil Sharpener.
7:
8:    As a result of your accident, we
9:    are redesigning our manual
10:   to warn our customers against trying
11:   to sharpen metal objects.
12:
13:   We hope that you will be more
14:   careful with electrical appliances
15:   in the future.
16:
17:   Sincerely,
18:
19:   *I.R. Sharpe, President
```

But now, say you decide not to warn Mr. Dimm about being careful. To delete lines 12 through 15, type the following:

```
12,15d
```

The result of this command is:

```
1:    Dear Mr. Dimm,
2:
3:    I was sorry to hear of your recent
4:    hospitalization due to electrical
5:    shock from our X-1000 Automatic
6:    Pencil Sharpener.
7:
8:    As a result of your accident, we
9:    are redesigning our manual
10:   to warn our customers against trying
11:   to sharpen metal objects.
12:
13:   Sincerely,
14:
15:   *I.R. Sharpe, President
```

To close up the space you might decide to delete line 7. You could just type the command **7d**. The result would be:

```
1:    Dear Mr. Dimm,
2:
3:    I was sorry to hear of your recent
4:    hospitalization due to electrical
5:    shock from our X-1000 Automatic
6:    Pencil Sharpener.
7:    *As a result of your accident, we
8:    are redesigning our manual
9:    to warn our customers against trying
10:   to sharpen metal objects.
11:
12:   Sincerely,
13:
14:   I.R. Sharpe, President
```


Suppose, though, that you just want a quick message that does not take responsibility for the accident. To delete a range of lines beginning with the current line, line 7, through line 11, type the following:

,11d

The result of this command is:

```
1:    Dear Mr. Dimm,
2:
3:    I was sorry to hear of your recent
4:    hospitalization due to electrical
5:    shock from our X-1000 Automatic
6:    Pencil Sharpener.
7:
8:    Sincerely,
9:
10:   I.R. Sharpe, President
```

Notice that as you delete lines, **edlin** automatically renumbers the remaining lines in the file.

edit

The **edit** command edits a line of text.

Syntax

[*line no.*]

Description

The *line no.* option specifies the line of text you want to edit. When you type a line number as a command, **edlin** displays the line number and the text on that line; then, on the line below, **edlin** reprints the line number. Now you can retype the line, or use the **edlin** editing keys to edit it. The existing text of the line serves as the template until you press the <Return> key.

If you do not type a line number (that is, if you only press the <Return> key), **edlin** edits the line after the current line. (The current line is shown by an asterisk (*).)

If you do not need to change the current line and if the cursor is at the beginning or end of the line, you can simply press the <Return> key to accept the line.

If you press the <Return> key while the cursor is in the middle of a line, **edlin** deletes the remainder of the line.

Examples

Suppose that the following file exists and is ready to edit:

```
1:  Dear Mr. Dimm,
2:
3:  I was sorry to hear of your recent
4:  hospitalization due to electrical
5:  shock from our Automatic
6:  Pencil Sharpener.
```

In line 5, say you want to insert the product's name, X-1000. To edit line 5, type **5**. **edlin** then displays the contents of the line with the cursor below the line:

```
5:  *shock from our Automatic
5:  *_
```

Now type the following command:

<F2> A INS X-1000

The **edlin** program skips to the *A* in *Automatic* and inserts *X-1000*, changing line 5 to:

```
5:  *shock from our X-1000
```

Then, press the <F3> key and <Return>, and you get:

```
5:  *shock from our X-1000 Automatic
```

At the **edlin** prompt, type **L** to see the file:

```
1:  Dear Mr. Dimm,
2:
3:  I was sorry to hear of your recent
4:  hospitalization due to electrical
5:  *shock from our X-1000 Automatic
6:  Pencil Sharpener.
```

end

The **end** command ends the editing session.

Syntax

e

Description

The **e** (end) command saves the edited file on your disk, renames the original input file *filename.bak*, and then exits **edlin**. If you created the file during the current editing session, **edlin** does not create a backup (*.bak*) file.

The **e** command takes no options. This means that you must select the drive that you want to save the file on when you start **edlin**; if you do not, **edlin** saves the file on the disk in the default drive. However, you can still copy the file to a different drive by using the MS-DOS **copy** command.

Before using the **e** command to save your file, make sure that the disk contains enough free space for the entire file. If it does not, **edlin** may not be able to write the entire file to the disk. The edited file will be lost, even though **edlin** may have saved part of it on the disk.

Examples

To end your editing session, type **e** and then press <Return>. After **edlin** processes the **e** command, the screen displays the MS-DOS prompt (for example, A>).

insert

The **insert** command inserts text immediately before the specified line.

Syntax

[*line*]**i**

Description

If you are creating a new file, you must type the **i** (insert) command before you can insert a new line of text. Text begins on line 1, and each time you press the <Return> key the next line number appears automatically.

edlin remains in insert mode until you press <CTL>**c**. When you finish the insertion and exit insert mode, the line immediately following the inserted lines becomes the current line. **edlin** automatically increments the line numbers that follow the inserted section by the number of lines that you inserted.

If you do not specify *line*, the default is the current line number and **edlin** inserts the lines before the current line. If *line* is a number larger than the last line number, or if you specify a cross hatch (#) as *line*, **edlin** appends the inserted lines to the end of the file. In this case, the last line that you inserted becomes the current line.

Examples

Suppose the following file exists and is ready to edit:

```
1:    Dear Mr. Dimm,
2:
3:    I was sorry to hear of your recent
4:    hospitalization due to electrical
5:    shock from our X-1000 Automatic
6:    Pencil Sharpener.
7:
8:    Sincerely,
9:
10:   I.R. Sharpe, President
```

Since this letter does not really offer any compensation for the accident, you might decide to add a brief note about a small gift that you are enclosing for Mr. Dimm. To insert text before line 8, type **8i**. The result is:

```
8:  *_
```

Now type the following lines, which will begin on line 8:

```
8:  *As a result of your accident, we
```

Press the <Return> key at the end of each line, and continue typing the next line:

```
9:  *are redesigning our manual to
10: *warn our customers against trying
11: *to sharpen metal objects.
```

To end the insertion, press <CTL>**c** on the next line, and then type **L** to list the file. The result is:

```
1:  Dear Mr. Dimm,
2:
3:  I was sorry to hear of your recent
4:  hospitalization due to electrical
5:  shock from our X-1000 Automatic
6:  Pencil Sharpener.
7:
8:  As a result of your accident, we
9:  are redesigning our manual to
10: warn our customers against trying
11: to sharpen metal objects.
12: *Sincerely,
13:
14: I.R. Sharpe, President
```

To insert a blank line immediately before the current line (line 12), type **i**. The result is:

```
12: *_
```

Insert a blank line by pressing <Return>, and end the insertion by pressing <CTL>**c** on the next line. Then, to list the file and see the result, type **L**.

```
1:    Dear Mr. Dimm,
2:
3:    I was sorry to hear of your recent
4:    hospitalization due to electrical
5:    shock from our X-1000 Automatic
6:    Pencil Sharpener.
7:
8:    As a result of your accident, we
9:    are redesigning our manual to
10:   warn our customers against trying
11:   to sharpen metal objects.
12:
13:   *Sincerely,
14:
15:   I.R. Sharpe, President
```

To append new lines to the end of the file, type **16i**. This produces the following:

```
16:   *_
```

Now type the following new lines:

```
16:   Sharpe Office Supplies
17:   The World Leader in Office Sharpeware
18:   Our motto: "You oughta be Sharpe too"
```

To get out of insert mode, press <CTL>**c** on line 19. The new lines will appear at the end of all previous lines in the file. Now list all the lines by typing **1,23L**. The result is:

```
1:    Dear Mr. Dimm,
2:
3:    I was sorry to hear of your recent
4:    hospitalization due to electrical
5:    shock from our X-1000 Automatic
6:    Pencil Sharpener.
7:
8:    As a result of your accident, we
9:    are redesigning our manual to
10:   warn our customers against trying
11:   to sharpen metal objects.
12:
13:   Sincerely,
14:
15:   I.R. Sharpe, President
16:   Sharpe Office Supplies
17:   The World Leader in Office Sharpeware
18:   Our motto: "You oughta be Sharpe too"
```

list

The **list** command lists a range of lines, including the two lines specified.

Syntax

[*line*][,*line*]**L**

Description

A capital letter "L" has been used here to avoid confusion with the number one. A small letter "l" would work just as well.

edlin provides default values if you do not type either option. If you do not type the first *line* option, as in the following command, the display will start 11 lines before the current line and end with the specified line:

,*line* **L**

The beginning comma is needed to show that you omitted the first *line* option.

Note

If the specified line is more than 11 lines before the current line, the display will be the same as if you omitted both options.

---

edlin displays 23 lines, starting with the specified line, if you omit the second option, as in the following command:

line **L**

If you just type **L**, **edlin** displays 23 lines – the 11 lines before the current line, the current line, and the 11 lines after the current line. If there are less than 11 lines before the current line, **edlin** displays more than 11 lines after the current line, up to a total of 23 lines.

Examples

```
1:    Dear Mr. Dimm,
2:
3:    I was sorry to hear of your recent
4:    hospitalization due to electrical
5:    shock from our X-1000 Automatic
6:    Pencil Sharpener.
.
.
.
15:   We also intend to attach a metal
16:   detector to future models.
.
.
.
26:   I.R. Sharpe, President
27:   Sharpe Office Supplies
28:   The World Leader in Office Sharpeware
29:   Our motto: "You oughta be Sharpe too"
```

To list a range of lines without referring to the current line, type **2,6L**. The result is:

```
2:
3:    I was sorry to hear of your recent
4:    hospitalization due to electrical
5:    shock from our X-1000 Automatic
6:    Pencil Sharpener.
```

To list a range of lines beginning with the current line, type **16,L** or **,L**. The result is:

```
16:   *detector to future models.
.
.
.
26:   I.M. Sharpe, President
27:   Sharpe Office Supplies
28:   The World Leader in Office Sharpeware
29:   Our motto: "You oughta be Sharpe too"
```

To list a range of 23 lines centered around the current line, type **L**. The result is:

```
5:     shock from our X-1000 Automatic
6:     Pencil Sharpener.
.
.
.
15:    We also intend to attach a metal
16:    *detector to future models.
.
.
.
26:    I.R. Sharpe, President
27:    Sharpe Office Supplies
```

move

The **move** command moves a range of text to the specified line.

Syntax

[*line*],+*line*,*line***m**

Description

The **M** (move) command lets you move a block of text to another location in a file. The first and second *line* options specify the range of lines that you want to move. The third *line* option specifies the line you want to move the first line in the range to.

edlin automatically renumbers the lines after it moves them. For example, the following command moves the text from the current line – plus 25 lines – to line 100:

,+25,100m

If the line numbers that you specify overlap, **edlin** displays an "Entry error" message.

Examples

Suppose the following file exists and is ready to edit.

```
1:    Dear Mr. Dimm,
2:
3:    I was sorry to hear of your recent
4:    hospitalization due to electrical
5:    shock from our X-1000 Automatic
6:    Pencil Sharpener.
7:
8:    As a result of your accident, we
9:    are redesigning our manual to
10:   warn our customers against trying
11:   to sharpen metal objects.
12:
13:   Sincerely,
14:
15:   I.R. Sharpe, President
```

```
16:   Sharpe Office Supplies
17:   The World Leader in Office Sharpeware
18:   Our motto: "You oughta be Sharpe too"
```

What if you prefer to have the motto at the start of the letter? If so, you could move lines 16 – 18 to line 1 by typing the following command:

16,+2,1m

The result of this command is:

```
1:    Sharpe Office Supplies
2:    The World Leader in Office Sharpeware
3:    Our motto: "You oughta be Sharpe too"
4:    Dear Mr. Dimm,
5:
6:    I was sorry to hear of your recent
7:    hospitalization due to electrical
8:    shock from our X-1000 Automatic
9:    Pencil Sharpener.
10:
11:   As a result of your accident, we
12:   are redesigning our manual to
13:   warn our customers against trying
14:   to sharpen metal objects.
15:
16:   Sincerely,
17:
18:   I.R. Sharpe, President
```

page

The **page** command displays a file one page (23 lines) at a time.

Syntax

[line][,line]**p**

Description

The first *line* option specifies the line at which **edlin** starts displaying. The second *line* option specifies how many lines appear on each page. If you do not type the first line, **edlin** starts the page at the line after the current line. If you do not type the second *line* option, **edlin** lists 23 lines on each page.

quit

The **quit** command quits the editing session, does not save any editing changes, and exits to the MS-DOS operating system.

Syntax

q

Description

This command is useful if you do not want to make any changes to a file. If you use the **q** (quit) command, **edlin** prompts you to make sure you do not want to save the changes. If you do want to save your changes, you should refer to the **e** (end) command for more information.

If you want to quit the editing session, type **y** (for Yes) when prompted; remember, though, that **edlin** will not save your editing changes and will not create a backup (*.bak*) file. Refer to the **e** command in this chapter for information about the *.bak* file.

If you want to continue the editing session, type **n** (for No).

When you exit **edlin**, it erases any previous copy of the file that has a *.bak* extension. If you reply **y** to the "Abort edit (Y/N)?" message, **edlin** deletes your previous backup copy.

Examples

To quit **edlin** without saving your changes, first type **q**. You will get the following message:

```
Abort edit (Y/N)?
```

When the message appears, type **y** and press <Return>.

replace

The **replace** command replaces all occurrences of a string of text in a range with a different string of text.

Syntax

[*line*][,*line*][?]**r***text1* <CTL>z *text2*

Description

The first and second *line* options specify the range of lines that the **R** (replace) command uses. Each time **edlin** finds *text1*, it replaces it with *text2*, displaying each line that changes. If a line contains two or more replacements, it is displayed once for each change. When **edlin** has made all the changes, the **r** command ends and the asterisk prompt reappears.

If you want to replace one string of text with another, you must separate the two with a <CTL>**z**. You can end the second string by pressing the <Return> key.

If you do not specify *text1*, the **r** command assumes the old (the previous) value. If this is the first replacement that you have made during the current editing session, and if you do not specify *text1*, the command ends. If you do not specify *text2*, you must end *text1* by pressing the <Return> key.

If you omit the first *line* option, **edlin** uses the line after the current line, by default. The default for the second *line* option is #. Remember that # refers to the line after the last line of the file.

Examples

If you end *text1* with a <CTL>**z** and do not specify *text2*, **edlin** assumes you want blank spaces for *text2*. For example, suppose you want to delete all occurrences of the word *clients* from your file. To do this you could simply type the following command, then press <CTL>**z** and <Return>:

```
rclients
```

The next command replaces clients with the previous *text2*:

```
rclients
```

The following command makes the previous *text1* become the previous *text2*:

r

Note that "previous" refers to an earlier string of text specified in an **s** or **r** command.

If you specify a question mark (**?**) in the **r** command, **edlin** stops at each line with text that matches *text1*, displays the line with *text2*, and then displays the prompt "O.K.?" If you press **y** or <Return>, *text2* replaces *text1*, and **edlin** looks for the next occurrence of *text1*. It continues this process until it reaches the end of the range or the end of the file. After **edlin** finds the last *text1*, it displays the asterisk prompt.

If you press any key other than **y** or the <Return> key after the "O.K.?" prompt, *text1* is left as is, and **edlin** moves to the next occurrence of *text1*. If *text1* occurs more than once in a line, **edlin** individually replaces each occurrence of *text1*, and displays the "O.K.?" prompt after each replacement. In this way you can replace only the desired *text1* strings and prevent unwanted substitutions.

Suppose the following file exists and is ready for editing:

```
1:    Dear Mr. Dimm,
2:
3:    I was sorry to hear of your recent
4:    hospitalization due to electrical
5:    shock from our X-1000 Automatic
6:    Pencil Sharpener.
7:
8:    As a result of your accident, we
9:    are redesigning our manual to
10:   warn our customers against trying
11:   to sharpen metal objects.
12:
13:   Sincerely,
14:
15:   I.R. Sharpe, President
```

Now, suppose that in lines 5 through 10 you want to replace all occurrences of the word *our* with the word *the*. To do this you would simply type **5,10 rour**, press <CTL>**z**, type **the**, and press the <Return> key. The result is:

```
5:    shock from the X-1000 Automatic
8:    As a result of ythe accident,
9:    are redesigning the manual to
10:   warn the customers against trying
```

In the previous example, some unwanted changes occurred. To avoid these and to confirm each replacement, you can use the same file with a slightly different command.

In the next example (assuming you are starting with the original file once again, ready for editing) you will see how to replace only certain occurrences of *our* with *the*. At the **edlin** prompt, type the following sequence of keys and words, and then press the <Return> key:

1,15? rour <CTL>z the

The result is:

```
5:    shock from the X-1000 Automatic
O.K.? y

8:    As a result of ythe accident, we
O.K.? n

9:    are redesigning the manual to
O.K.? y

10:   warn the customers against trying
O.K.? n

*_
```

Type the list command, **L**, to see the result of all these changes:

```
.
.
5:      shock from the X-1000 Automatic
.
.
8:      As a result of your accident, we
9:      are redesigning the manual to
10:     warn our customers against trying
.
.
```

search

The **search** command searches a range of lines for a string of text.

Syntax

[*line*][,*line*][?]s*text*

Description

The first and second *line* options specify the range of lines for **edlin** to search. You end the *text* option by pressing the <Return> key. **edlin** displays the first line that matches the string; that line then becomes the current line. Unless you type the question mark (**?**) option, the **S** (search) command ends when it finds the first match. If **edlin** cannot find a line with a match, it displays the message "Not found."

If you include the question mark (**?**) option, **edlin** displays the first line with matching text and prompts you with the message "O.K.?" If you press either **y** (Yes) or the <Return> key, this line becomes the current line and the search ends. If you press any other key, the search continues until another match is found, or until all lines have been searched. (The search ends when **edlin** displays the "Not found" message.)

If you do not type the first line number, **edlin** defaults to the line after the current line; if you do not type the second line number, it defaults to # (the line after the last line of the file).

If you omit the *text* option, **edlin** uses the text from the previous **S** or **R** (replace) command. If this is the first **s** or **r** command you have used during the current session, and you have not specified a search string, the **s** command ends immediately.

Examples

Suppose the following file exists and is ready for editing:

```
1:    Dear Mr. Dimm,
2:
3:    I was sorry to hear of your recent
4:    hospitalization due to electrical
5:    shock from our X-1000 Automatic
6:    Pencil Sharpener.
7:
8:    As a result of your accident, we
9:    are redesigning our manual to
10:   warn our customers against trying
11:   to sharpen metal objects.
12:
13:   Sincerely,
14:
15:   I.R. Sharpe, President
```

To search for the first occurrence of the word *to*, type the following command, then press the <Return> key:

2,12 sto

edlin displays the following lines:

```
3:    I was sorry to hear of your recent
```

To get the *to* in line 4, modify the **s** command by pressing <DEL>, <F3>, <Return>. The search continues at the line after the current line (i.e., the search starts at line 4), because you deleted the first line number in the template. The result is:

```
4:    hospitalization due to electrical
```

To search through several occurrences of a string until the correct string is found, type the command **1, ? sto** The result is:

```
3:    I was sorry to hear of your recent
O.K.? _
```

If you press any key (except **y** or the <Return> key), the search continues, so type **n** here:

```
O.K.?  n
```

Continue:

```
4:    hospitalization due to electrical
O.K.? _
```

Now press **y** to terminate the search:

```
O.K.?  y
*_
```

edlin reports a match and continues to search for the same string when you retype the **s** command and press <Return>:

```
s
```

edlin reports another match, if there is one.

```
s
```

edlin displays the "Not found" message at the end of the search.

Note that the text string defaults to any string specified by a previous **r** or **s** command.

transfer

The **transfer** command inserts, at a specific line number, the contents of another file into the file that you are currently editing.

Syntax

[*line*] **t** *filename*

Description

The **t** (transfer) command puts the contents of one file into another file, or into the text you are typing. **edlin** inserts *filename* at the line number you give in the *line* option, and then automatically renumbers the lines. If you omit the line number, **edlin** inserts the text, beginning on the current line.

Example

To copy a file named *irsharpe.mem* to line 12 of the file you are editing, use the following command:

```
12 t irsharpe.mem
```

write

The **write** command writes a specific number of lines to a disk.

Syntax

[*n*]w

Description

The *n* option specifies the number of lines that you want to write to the disk. You need this command only if the file you are editing is too large to fit into memory. When you start **edlin**, it reads lines from your file until memory is ¾ full.

To edit the remainder of your file, you must write the edited lines in memory to your disk. Then you can load additional unedited lines from your disk into memory by using the **A** (append) command, which is described earlier in this chapter.

Note

If you do not specify the number of lines for **edlin** to write, it writes lines until memory is ¾ full. But it does not write any lines to your disk until memory is more than ¾ full. Also, **edlin** renumbers all of the lines so that the first remaining line becomes line number 1.

Chapter 8

link: A Linker

Introduction

In this chapter you will learn:

- how to create executable files with **link**
- how to use **link** command options
- how **link** creates programs.

The Microsoft 8086 Object Linker (**link**) creates executable programs from object files generated by the Microsoft Macro Assembler (MASM) or by compilers for high-level languages, such as C or Pascal. The linker copies the resulting program to an executable (*.exe*) output file. You can then run the program by typing the file's name at the MS-DOS prompt.

To use **link**, you must create one or more object files, then submit these files, along with any required library files, to the linker for processing. **link** combines code and data in the object files and searches the named libraries to resolve external references to routines and variables. It then copies a relocatable execution image and the relocation information to the executable file. Using the relocation information, MS-DOS can load the executable image at any convenient memory location and then run it. **link** can process programs that contain up to one megabyte of code and data.

This chapter explains how to use the linker to create executable programs. It also defines each of the options you can use in a **link** command line to control the linking process. Finally, it explains how **link** creates programs.

Using the Linker

This section explains three methods for starting and using the linker to create executable programs. These methods let you use **link** by answering a series of prompts, by typing an MS-DOS command line, or by using a response file. The three methods can also be mixed.

Once you start **link**, it will either process the files you supply or prompt you for additional files. You can stop the linker at any time by pressing <CTL>c.

Using Prompts to Specify Link Files

When you type **link** at the MS-DOS prompt, the linker prompts you for the information it needs. Follow these steps:

1. First, type the following command and press the <Return> key:

   link

   link prompts you for the object files you wish to **link** by displaying the following message:

   ```
   Object Modules [.OBJ]:
   ```

2. Type the name or names of the object files you wish to link. If you do not supply extensions for these files, **link** supplies *.obj* by default. If you have more than one name, make sure you separate each with spaces or plus signs (+). If you have more names than can fit on one line, type a plus sign (+) as the last character on the line and press the <Return> key, **link** then prompts you for additional object files.

Once you have given all your object filenames, press the <Return> key. The linker displays the following prompt:

```
Run File [filename.EXE]:
```

Note

In step 2, *filename* is the same as the first filename you typed at the "Object Modules" prompt.

3. Type the name of the executable file you wish to create, and press the <Return> key. If you do not give an extension, **link** supplies *.exe* by default. If you want **link** to supply a default executable filename, just press the <Return> key. The filename will then be the same as the first object file, but with the extension *.exe*.

   Once you have pressed the <Return> key, **link** displays the following prompt:

```
List File [NUL.MAP]:
```

4. Type the name of the map file you wish to create, then press the <Return> key. If you do not supply a filename extension, the linker uses *.map* by default. If you don't want a map file, don't type a filename, just press the <Return> key.

   Once you have pressed the <Return> key, **link** displays the following prompt:

```
Libraries [.LIB]:
```

5. Type the names of any library files containing routines or variables referenced but not defined in your program. If you give more than one name, make sure the names are separated by spaces or plus signs (+). If you don't supply filename extensions, the linker uses *.lib* by default. If you have more names than can fit on one line, type a plus sign (+) as the last character on the line and press the <Return> key. **link** then prompts you for additional filenames.

   After entering all names, press the <Return> key. If you don't want to search any libraries, don't type any names; just press the <Return> key.

6. **link** now creates the executable file.

When entering filenames, you must supply a pathname for any file that is not in the current drive and directory. You can use **link** options by typing them after the filename at any prompt. If the linker cannot find an object file, it displays a message and waits for you to change disks, if necessary.

At any prompt, you can type the rest of the filenames by using the command line format described in the next section, "Using a Command Line to Specify Link Files." For example, you can choose the default responses for all remaining prompts by typing a semicolon (;) after any prompt, or you can type commas (,) to indicate several files. (If you type a semicolon at the "Object Modules" prompt, be sure to supply at least one object filename.) When the linker encounters a semicolon, it immediately chooses the default responses and processes the remaining files without displaying any more prompts.

Example

The following example does the following:

- Links the object modules *moda.obj*, *modb.obj*, *modc.obj*, and *startup.obj*;
- searches the library file *math.lib* on drive B: of the *\lib* directory for routines and data used in the program; and
- creates an executable file named *moda.exe*, and a map file named *abc.map*.

The **/pause** option in the Object Modules prompt line then causes **link** to pause while you change disks, after which the linker creates the executable file:

```
Object Modules [.OBJ]: moda+modb+
Object Modules [.OBJ]: modc+startup/PAUSE
Run File [moda.EXE]:
List File [NUL.MAP]: abc
Libraries [.LIB]: b:\lib\math
```

Using a Command Line to Specify Link Files

You can create an executable program by typing **link**, followed by the names of the files you wish to process. The command line has the following general form:

link *objectfiles* [,[*executablefile*] [,[*mapfile*] [,[*libraryfiles*]]]]
[*options*] [;]

The variables in this command line are described as follows:

objectfiles includes the name or names of object files that you want to link together. The files must have been created using MASM or a high-level-language compiler. The linker requires at least one object file. If you do not supply an extension, link provides the extension *.obj*.

executablefile is an optional placeholder for the name you wish to give the executable file that **link** will create. If you do not supply an *executablefile*, **link** creates a filename by using the name of the first object file in the command line and appending it with an *.exe* extension.

mapfile is the name of the file that receives the map listing. If you do not supply an extension, the linker provides the default extension *.map* which is appended to the first object file specified on the command line. If you specify the **/map** or **/linenumbers** option, the linker creates a map file even if you don't specify one in your command line.

libraryfiles includes the name or names of the libraries containing routines that you wish to link to create a program. If you do not supply an extension, **link** supplies the extension *.lib*.

options controls the operation of link. You can use any of the options listed in the section entitled "The link Options." You can specify options anywhere on the command line.

The commas (,) you use to separate filenames for the different types of files are required even if you don't supply a filename. If you want the filename for a file to be the default (the same as the base name of the first object file), you can type the comma that would follow the filename without actually supplying a filename. If you type the comma after the object file, **link** supplies the default name for the *executablefile* and suppresses the *mapfile* and *libraryfiles*. You can also use a semicolon (;) anywhere after the object file to terminate the command line.

If you do not supply all filenames in the command line and do not end the command line with a semicolon, the linker prompts you for additional files, using the prompts described previously in the section, "Using Prompts to Specify Link Files." If you give more than one object file or library file, you must separate the names by spaces or plus signs (+).

If you do not specify a drive or directory for a file, **link** assumes the file is on the current drive and directory. You cannot specify the drive or directory for the *objectfile* and expect **link** to supply the same drive and directory for other files. Instead, you must give the location of each file specifically.

Note

When linking modules produced by a high-level-language compiler that supports overlays, you must specify overlay modules by putting them in parentheses. Since MASM has no overlay manager, you can specify overlays only for object files linked with the runtime library of a language compiler that supports overlays.

For example, you can use overlays with modules compiled with Microsoft FORTRAN, versions 3.2 and later, Microsoft Pascal versions 3.2 and later, and Microsoft C versions 3.0 and later. See your language compiler's manual for details on specifying overlays.

Examples

The first example below uses the object file *file.obj* to create the executable file *file.exe*. **link** searches the *routine.lib* library for routines and variabl program. It also creates a file called *file.map*, which contains a list of the program's segments and groups:

link file.obj,file.exe,file.map,routine.lib

The first example is equivalent to the following line:

link file,,,routine

The second example uses the two object files, *startup.obj* and *file.obj*, on the current drive to create an executable file named *file.exe* on drive B. **link** creates a file called *file.map* on the *\map* directory of the current drive, but does not search any libraries:

link startup+file,b:file,\map\file;

The final example links the object modules *moda.obj*, *modb.obj*, *modc.obj*, and *startup.obj*:

link moda modb modc startup/PAUSE,,abc,b:\lib\math

The linker searches through the library file *math.lib* in the *\lib* directory on drive B: for routines and data used in the program. It then creates an executable file named *moda.exe*, and a map file named *abc.map*.

The **/PAUSE** option in the command line causes the linker to pause and ask you to change disks before it creates the executable file.

Using a Response File to Specify Link Files

You can create a program by listing, in a response file, the names of all the files to be processed, and by giving the name of the response file on the **link** command line. The simplest way to use a response file is with a command line of the following form:

link @*filename*

You can also specify a response file at any prompt, or at any position in a command line. The input from the response file is treated exactly as though you had typed it at the **link** prompts or in a command line. However, any carriage return/linefeed combinations in the file are treated as if

you had pressed the <Return> key in response to a prompt, or typed a comma in a command line.

When you specify a response file, remember that the filename must be the name of the response file, and that you must precede it by an "at" sign (@). If the file is in another directory or on another disk drive, you must provide a pathname.

The Format of a Response File

You can name the response file anything you like. The file content has the following general form:

objectfiles
[*executablefile*]
[*mapfile*]
[*libraryfiles*]

You can omit any elements that have already been provided at prompts or with a partial command line.

You must place each group of filenames on a separate line. If you have more names than can fit on one line, you can simply continue the names on the next line by typing a plus sign (+) as the last character in the current line and pressing the <Return> key. If you do not supply a filename for a group, you must leave an empty line. You can give options on any line.

You can place a semicolon (;) on any line in the response file. When **link** encounters the semicolon, it automatically supplies default filenames for all files you have not yet named in the response file. The remainder of the response file is ignored.

When you create a program with a response file, the linker displays each response from your response file on the screen in the form of prompts. If the response file does not contain names for required files, **link** prompts you for the missing names and waits for you to enter responses.

Note

A response file should end with either a semicolon (;) or a carriage return/linefeed combination. If you fail to provide a final carriage return/linefeed in the file, the linker will display the last line of the response file and wait for you to press the <Return> key.

Example

The following response file tells the linker to link the four object modules, *moda*, *modb*, *modc*, and *startup*. Then, before producing the executable file *moda.exe*, it tells **link** to pause to let you swap disks. Finally, the linker creates a map file *abc.map* and searches the *math.lib* library in the *\lib* directory of drive B:

```
moda modb modc startup /PAUSE

abc
b:\lib\math
```

The following procedure combines all three methods of supplying filenames. Assume you have a response file called *library* that contains one line:

```
lib1+lib2+lib3+lib4
```

Now start **link** with a partial command line:

link object1 object2

link takes *object1.obj* and *object2.obj* as its object files, and prompts for the next file:

```
Run File [object1.EXE]: exec
List File [NUL.MAP]:

Libraries [.LIB]: @library
```

You include the name *exec* so that the linker will name the executable file *exec.exe*. You then press the <Return> key to indicate that no map file is desired, and you enter **@library** so that the linker will read in the response file containing the four library filenames.

Using Search Paths with Libraries

You can direct **link** to search directories and disk drives for the libraries you have named in a command by either specifying one or more search paths with the library names, or by assigning the search paths to the environment variable LIB before you invoke **link**. See Appendix E, "MS-DOS Commands".

A search path is the path of a directory or drive name. You type search paths along with library names on the **link** command line or in response to the "Libraries" prompt. You can also specify up to 16 search paths and assign them to the LIB environment variable by using the MS-DOS **set** command. In the latter case, you must separate the search paths by semicolons (;).

If you include a drive or directory name in the filename for a library in the **link** command line, the linker searches there only. If you don't give a drive or directory name, **link** searches for library files in the following order:

1. First, the linker searches the current drive and directory.
2. If the library is not found and one or more search paths have been given in the command line, the linker searches the specified search paths in the order in which you gave them.
3. If the library is still not found and you have set a search path by using the LIB environment variable, the linker searches there.
4. If the library is still not found, **link** prints an error message.

Examples

In the first example, the linker searches only the *\altlib* directory on drive A: to find the *math.lib* library. To find *common.lib*, it will search the current directory on the current drive, the current directory on drive B:, and finally the *\lib* directory on drive D:

```
link file,,file,A:\altlib\math.lib+common+B:+D:\lib\
```

In the second example, **link** searches the current directory, the *\lib* directory on drive C:, and the *\system\lib* directory on drive U: to find the libraries, *math.lib* and *common.lib*:

```
set LIB=C:\lib;U:\system\lib
link file,file.map,math+common
```

The Map File

The map file lists the names, load addresses, and lengths of all segments in a program. It also lists the names and load addresses of any groups in the program, the program start address, and messages about any errors it may have encountered. If the **/map** option is used in the **link** command line, the map file lists the names and load addresses of all public symbols.

Segment information has the general form shown in this example:

```
Start         Stop          Length        Name   Class

00000H        0172CH        0172DH        TEXT   CODE
01730H        01E19H        006EAH        DATA   DATA
```

The Start and Stop columns show the 20-bit addresses (in hexadecimal) of the first and last byte in each segment. These addresses are relative to the beginning of the load module, which is assumed to be address 00000H. The operating system chooses its own starting address once the program is actually loaded. The Length column gives the length of the segment in bytes; the Name column gives the name of the segment; and the Class column gives the segment's class name.

Group information has the following general form:

```
Origin        Group

0000:0        IGROUP
0173:0        DGROUP
```

In this example, IGROUP is the name of the code (instruction) group and DGROUP is the name of the data group.

At the end of the listing file, the linker gives you the address of the program entry point.

If you specify the **/map** option in the **link** command line, the linker adds a public-symbol list to the map file. The symbols are presented twice: once in alphabetical order, then in the order of their load addresses. The list has the general form shown in the following example:

```
Address Publics by Name

0000:1567    BRK
0000:1696    CHMOD
0000:01DB    CHKSTK
0000:131C    CLEARERR
0173:0035    FAC

Address Publics by Value

0000:01DB    CHKSTK
0000:131C    CLEARERR
0000:1567    BRK
0000:1696    CHMOD
0000:0035    FAC
```

The addresses of the public symbols are in *segment:offset* format. They show the location of the symbol relative to the beginning of the load module, which is assumed to be at address 0000:0000.

When the **/high** and **/dsallocate** options are used and the program's code and data combined do not exceed 64K bytes, the map file may show symbols that have unusually large segment addresses. These addresses indicate a symbol whose location is below the actual start of the program code and data.

For example, the following symbol entry shows that TEMPLATE is located below the start of the program:

```
FFF0:0A20 TEMPLATE
```

The 20-bit address of TEMPLATE is 00920H.

The Temporary Disk File – vm.tmp

link normally uses available memory for the link session. If it runs out of available memory, it creates a temporary disk file named *vm.tmp* in the current working directory. When the linker creates this file, it displays the following message:

```
VM.TMP has been created.
Do not change diskette in drive x:
```

After this message appears, you must not remove the disk from the drive specified by *x* until the **link** session ends. The **/pause** option cannot be used if a temporary file is created. After **link** has created the executable file, it deletes the temporary file automatically.

Note

Do not use the *vm.tmp* filename for your own files, since when the linker creates the temporary file, it destroys any previous file that has the same name.

The link Options

The linker options specify and control the tasks that **link** performs. All options begin with the linker-option character, which is a slash (/). You can use the following options anywhere on a **link** command line:

Option name	Action
/help	Shows the list of options
/pause	Pauses during linking
/exepack	Packs executable files
/map	Creates a public-symbol map
/linenumbers	Copies line numbers to a map file
/noignorecase	Preserves case sensitivity in names
/nodefaultlibrarysearch	Overrides default libraries
/stack	Sets maximum allocation space
/cparmaxalloc	Sets maximum number of paragraphs required by program
/high	Sets a high load address for a program
/dsallocate	Allocates a data group
/nogroupassociation	Sets a group association override
/overlayinterrupt	Sets an overlay interrupt
/segments	Sets a maximum number of segments
/dosseg	Specifies MS-DOS segment ordering

You can abbreviate an option name as long as your abbreviation contains enough letters to distinguish the specified option from other options. Minimum abbreviations are listed for each option.

Many of the **link** options set values in the MS-DOS program header.

/help

The **/help** option causes **link** to write a list of the available options to the screen. If you ever need a reminder of the available options, you may find this list convenient. You should not give a filename when using the **/help** option.

Minimum abbreviation: **/he**

Example

```
link /help
```

/pause

The **/pause** option causes **link** to pause before writing the executable file to disk so that you can swap disks before the linker writes the executable (*.exe*) file to disk.

If you specify the **/pause** switch, the linker displays the following message before creating the run file:

```
About to generate .EXE file
Change diskette in drive x: and press <ENTER>
```

x: is the proper drive name. This message appears after the linker has read data from the object files and library files, and after it has written data to the map file, if you specified one. **link** resumes processing when you press the <Return> key, and after it writes the executable file to disk, it displays the following message:

```
Please replace original diskette
in drive x: and press <ENTER>
```

Minimum abbreviation: **/p**

Note

Do not remove the disk used for the *vm.tmp* file, if such a file has been created. If the temporary disk message appears when you have specified the **/pause** option, you should press <CTL>**c** to terminate the **link** session. Rearrange your files so that the temporary file and the executable file can be written to the same disk, then try again.

Example

The following command causes the linker to pause just before creating the executable file *file.exe*. After creating this file, **link** pauses again to let you replace the original disk:

```
link file/pause,file,,\lib\math
```

/exepack

The **/exepack** option directs **link** to remove sequences of repeated bytes (typically nulls) and optimize the load-time relocation table before creating the executable file. Executable files linked with the **/exepack** option may be smaller and, thus, load faster than files linked without the option. However, the Microsoft Symbolic Debug Utility (**symdeb**) cannot be used with packed files.

The **/exepack** option does not always save a significant amount of disk space (in some cases it may even increase file size). Programs that have a large number of load-time relocations (about 500 or more) and long streams of repeated characters will usually be shorter if packed. If you are not sure if your program meets these conditions, try linking it both ways and compare the results.

Minimum abbreviation: **/e**

Example

This example creates a packed version of the file *program.exe*:

```
link program /e;
```

/map

The **/map** option causes **link** to produce a listing of all public symbols declared in your program. This list is copied to the map file that **link** creates. For a complete description of the listing-file format, see "The Map File" earlier in this chapter. The **/map** option is required if you want to use **symdeb** for symbolic debugging.

Minimum abbreviation: **/m**

If you do not specify a map file in a **link** command, you can use the **/map** option to force the linker to create one. **link** gives the forced map file the same filename as the first object file specified in the command. It also adds the default extension *.map*.

Example

The following command creates a map of all public symbols in the file *file.map*:

```
link file,,/map;
```

/linenumbers

The **/linenumbers** option directs the linker to copy the starting address of each program source line to a map file. The starting address is actually the address of the first instruction that corresponds to the source line. You can use the **mapsym** program to copy line-number data to a symbol file, which can then by used by **symdeb**.

The linker copies the line-number data only if you give a map-file name in the **link** command line, and only if the given object file has line-number information. Line numbering is available in some high-level-language compilers, including Microsoft FORTRAN and Pascal, versions 3.0 and later, and Microsoft C, versions 2.0 and later.

MASM does not copy line-number information to the object file. If an object file has no line-number information, the linker ignores the **/linenumbers** option.

Minimum abbreviation: **/li**

If you do not specify a map file in a **link** command, you can still use the **/linenumbers** option to force the linker to create one. Just place the option at or before the "List File" prompt. **link** gives the forced map file the same filename as the first object file that you specified in the command, and gives it the default extension *.map*.

Example

This example causes the line-number information in the object file *file.obj* to be copied to the map file *file.map*:

```
link file/linenumbers,,em+slibfp
```

/noignorecase

The **/noignorecase** option directs **link** to treat uppercase and lowercase letters in symbol names as distinct letters. Normally, **link** considers uppercase and lowercase letters to be identical, treating the words "TWO", "two", and "Two" as the same symbol. When you use the **/noignorecase** option, however, the linker treats "TWO", "two", and "Two" as different symbols.

Typically, you use the **/noignorecase** option with object files created by high-level-language compilers. Some compilers treat uppercase and lowercase letters as distinct letters and assume the linker does the same.

If you are linking modules created with MASM to modules created with a case-sensitive language such as C, make sure public symbols have the same sensitivity in both modules. For example, you could make all variables in C distinctive by spelling, regardless of case, and then link without the **/noignorecase** option. Another alternative would be to use the **/ML** or **MX** option to make public variables in MASM case-sensitive. Then link with the **/noignorecase** option.

Minimum abbreviation: **/noi**

Example

The following command causes the linker to treat uppercase and lowercase letters in symbol names as distinct letters. The object file *file.obj* is linked with routines from the standard C language library *libc.lib* located in the *\lib* directory. The C language expects uppercase and lowercase letters to be treated distinctly:

```
link file/noi,,,\lib\libc
```

/nodefaultlibrarysearch

The **/nodefaultlibrarysearch** option directs the linker to ignore any library names it may find in an object file. A high-level-language compiler may add a library name to an object file to ensure that a default set of libraries is linked with the program. Using this option overrides these default libraries and lets you explicitly name the libraries you want by including them on the **link** command line.

Minimum abbreviation: **/nod**

Example

The following example links the object files, *startup.obj* and *file.obj*, with routines from the libraries, *em*, *slibfp*, and *slibc*. Any default libraries that may have been named in *startup.obj* or *file.obj* are ignored:

```
link startup+file/nod,,,em+slibfp+slibc
```

/stack:*size*

The **/stack** option sets the program stack to the number of bytes given by *size*. The linker usually calculates a program's stack size automatically, basing it on the size of any stack segments given in the object files. If you do use the **/stack** option, the linker uses the value you type in place of any value it may have calculated.

The size can be any positive integer in the range from 1 to 65535. This value can be a decimal, octal, or hexadecimal number. Octal numbers must begin with a zero, and hexadecimal numbers must begin with a leading zero followed by a lowercase *x*: for example, 0x1B.

By using the **exemod** utility, you can also change the stack size after linking.

Minimum abbreviation: **/st**

Examples

The first example sets the stack size to 512 bytes:

```
link file/stack:512,,;
```

The second example sets the stack size to 255 (FFH) bytes:

```
link moda+modb,run/st:0xFF,ab,\lib\start;
```

The final example sets the stack size to 24 (30 octal) bytes:

```
link startup+file/st:030,,;
```

/cparmaxalloc:*number*

The **/cparmaxalloc** option sets the maximum number of 16-byte paragraphs needed by a program when it is loaded into memory. The operating system uses this number when allocating space for a program prior to loading it.

link normally sets the maximum number of paragraphs to 65535. Since this represents all addressable memory, the operating system always denies the default setting and allocates the largest contiguous block of memory it can find. If you use the **/cparmaxalloc** option, the operating system allocates no more space than is given by this option. This means any additional space in memory is free for other programs.

The number can be any integer in the range from 1 to 65535. It must be a decimal, octal, or hexadecimal number. Octal numbers must begin with a zero, and hexadecimal values must begin with a leading zero followed by a lowercase x. For example, 0x2B.

If number is less than the minimum number of paragraphs needed by the program, **link** ignores your request and sets the maximum value equal to the minimum needed. The minimum number of paragraphs needed by a program is never less than the number of paragraphs of code and data in the program.

Minimum abbreviation: **/c**

Examples

The first example sets the maximum allocation to 15 paragraphs:

```
link file/c:15,,;
```

The second example sets the maximum allocation to 255 (FFH) paragraphs:

```
link moda+modb,run/cparmaxalloc:0xff,ab;
```

The final example sets the maximum allocation to 24 (30 octal) paragraphs:

```
link startup+file,/c:030,;
```

/high

The **/high** option sets a program's starting address to the highest possible address in free memory. If you don't use the **/high** option, **link** sets the program's starting address as low as possible in memory.

Minimum abbreviation: **/h**

Example

This example sets the starting address of the program in *file.exe* to the highest possible address in free memory:

```
link startup+file/high,,;
```

/dsallocate

The **/dsallocate** option directs the linker to reverse its normal processing when assigning addresses to items belonging to the group named DGROUP. Normally, **link** assigns the offset 0000H to the lowest byte in a group. If you use **/dsallocate**, **link** assigns the offset FFFFH to the highest byte in the group. The result is data that appear to be loaded as high as possible in the memory segment containing DGROUP.

Typically, you use the **/dsallocate** option with the **/high** option to take advantage of unused memory before the start of the program. The linker assumes that all free bytes in DGROUP occupy the memory preceding the program. To use the group, you must set a segment register to the start address of DGROUP.

Minimum abbreviation: **/d**

Example

The following example directs the linker to place the program as high in memory as possible, then adjust the offsets of all data items in DGROUP so that they are loaded as high as possible within the group:

```
link startup+file/high/dsallocate,,,em+mlibfp
```

/nogroupassociation

The **/nogroupassociation** option directs **link** to ignore group associations when assigning addresses to data and code items.

Minimum abbreviation: **/nog**

This option exists strictly for compatibility with older versions of FORTRAN and Pascal (Microsoft versions 3.13 or earlier, or any IBM version prior to 2.0). You should never use the **/nogroupassociation** option except to link with object files produced by those compilers, or with the runtime libraries that accompany the old compilers.

/overlayinterrupt:*number*

The **/overlayinterrupt** option sets the interrupt number of the overlay loading routine to *number*. This option overrides the normal overlay interrupt number (03FH).

Number can be any integer value in the range from 0 to 255. It must be a decimal, octal, or hexadecimal number. Octal numbers must have a leading zero, and hexadecimal numbers must start with a leading zero followed by a lowercase x. For example, 0x3B.

MASM does not have an overlay manager. Therefore, you can use this option only if you are linking with a runtime module from a language compiler that supports overlays. Check your compiler documentation, since you may not be able to use this option with some compilers.

Minimum abbreviation: **/o**

Note

You should not use interrupt numbers that conflict with the standard MS-DOS interrupts.

Examples

The first example sets the overlay interrupt number to 255:

```
link file/o:255,,,87+slibfp
```

The second example sets the overlay interrupt number to 255 (FFH):

```
link moda+modb, run/overlay:0xff,ab.map,em+mlibfp
```

The final example sets the overlay interrupt number to 255 (377 octal):

```
link startup+file,/o:0377,,em+mlibfp
```

/segments:*number*

The **/segments** option directs the linker to process no more than *number* segments per program. If it encounters more than the given limit, the linker displays an error message, and stops linking. You use this option to override the default limit of 128 segments.

If you do not use **/segments**, the linker allocates enough memory space to process up to 128 segments. If your program has more than 128 segments, you will need to set the segment limit higher to increase the number of segments that **link** can process. If you get the following link error message, you should set the segment limit lower:

```
Segment limit set too high
```

The number can be any integer value in the range from 1 to 1024. It must be a decimal, octal, or hexadecimal number. Octal numbers must have a leading zero, and hexadecimal numbers must start with a leading zero followed by a lowercase x. For example, 0x4B.

Minimum abbreviation: **/se**

Examples

The first example sets the segment limit to 192:

```
link file/se:192,,;
```

The second example sets the segment limit to 255 (FFH).

```
link moda+modb,run/segments:0xff,ab,em+mlibfp;
```

/dosseg

The **/dosseg** option causes **link** to arrange all segments in the executable file according to the MS-DOS segment-ordering convention. This convention has the following rules:

- All segments having the class name CODE are placed at the beginning of the executable file.
- Any other segments that do not belong to the group, DGROUP, are placed immediately after the CODE segments.
- All segments belonging to DGROUP are placed at the end of the file.

If you do not use the **/dosseg** option, see "Order of Segments" later in this chapter for an explanation of the normal segment order.

Minimum abbreviation: **/do**

Example

The following command causes the linker to create an executable file, named *file.exe*, whose segments are arranged according to the MS-DOS segment-ordering convention. The segments in the object files *start.obj* and *test.obj*, and any segments copied from the libraries *math.lib* and *common.lib*, are arranged according to the same segment-ordering convention as above.

```
link start+test/dosseg,,,math+common
```

How link Works

link creates an executable file by concatenating a program's code and data segments according to the instructions in the original source files. These concatenated segments form an executable image that is copied directly into memory when you run the program. The order and manner in which the linker copies segments to the executable file defines the order and manner in which it loads the segments into memory.

You can tell the linker how to link a program's segments by using a SEGMENT directive to supply segment attributes, or by using the GROUP directive to form segment groups. These directives define group associations, classes, and align and combine types that define the order and relative starting addresses of all segments in a program. This information works in addition to any information you supply through command line options.

The following sections explain the process that **link** uses to concatenate segments and resolve references to items in memory.

Alignment of Segments

The linker uses a segment's align type to set the starting address for the segment. The align types are byte, word, para, and page. These types correspond to starting addresses at byte, word, paragraph, and page boundaries, representing addresses that are multiples of 1, 2, 16, and 256, respectively. The default align type is *para*.

When the linker encounters a segment, it checks the align type before copying the segment to the executable file. If the align type is *word*, *para*, or *page*, the linker checks the executable image to see if the last byte copied ends at an appropriate boundary. If it doesn't, **link** pads the image with extra null bytes.

Frame Number

The linker computes a starting address for each segment in a program. The starting address is based on a segment's align type and on the size of the segments already copied to the executable file. The address consists of an offset and a canonical frame number, which specifies the address of the first paragraph in memory that contains one or more bytes of the segment. A frame number is always a multiple of 16 (a paragraph address),

and the offset is the number of bytes from the start of the paragraph to the first byte in the segment. For *byte* and *word* align types, the offset may be nonzero, but the offset is always zero for *para* and *page* align types.

The frame number of a segment can be obtained from a **link** file. The frame number is the first five hexadecimal digits of the start address specified for the segment.

Order of Segments

link copies segments to the executable file in the same order that it encounters them in the object files. The linker maintains this order throughout the program unless it encounters two or more segments with the same class name. Segments with identical class names belong to the same class type, and are copied to the executable file as contiguous blocks.

Combined Segments

link uses combine types to determine whether two or more segments sharing the same name should be combined into a single large segment. The combine types are *public*, *stack*, *common*, *memory*, *at*, and *private*.

Public Combine Types

If a segment has a *public* combine type, the linker automatically combines it with any other segments that have the same name and belong to the same class. When **link** combines segments, it ensures that the segments are contiguous and that all addresses in the segments can be accessed using an offset from the same frame address. The result is the same as if the segment were defined as a whole in the source file.

The linker preserves each segment's align type. This means that even though the segments belong to a single, large segment, the code and data in the segments retain their original align type. If the combined segments exceed 64K bytes, **link** displays an error message.

Stack Combine Types

If a segment has a *stack* combine type, the linker carries out the same combine operation as for public segments. The only difference is that stack segments cause **link** to copy an initial stack-pointer value to the executable file. This stack-pointer value is the offset to the end of the first stack segment (or combined stack segment) that the linker encounters.

If you use the *stack* type for stack segments, you do not need to give instructions to load the segment into the SS register.

Common Combine Types

If a segment has a *common* combine type, the linker combines it automatically with any other segments that have the same name and belong to the same class. When **link** combines common segments, however, it places the start of each segment at the same address, creating a series of overlapping segments. The result is a single segment no larger than the largest of the combined segments.

Memory Combine Types

The linker treats segments with *memory* combine types exactly like segments with public combine types. The Microsoft Macro Assembler (MASM), provides combine type *memory* for compatibility with linkers that support a separate combine type for memory segments.

Private Combine Types

A segment has a *private* combine type only if no explicit combine type is defined for it in the source file. **link** does not combine *private* segments.

Groups

Groups permit noncontiguous segments that do not belong to the same class to be addressable relative to the same frame address. When **link** encounters a group, it adjusts all memory references to items in the group so that they are relative to the same frame address.

Segments in a group do not have to be contiguous and do not have to belong to the same class. Nor do they have to have the same combine type. The only requirement is that all segments in the group fit within 64K bytes.

Groups do not affect the order in which the segments are loaded. Unless you use class names and enter object files in the right order, there is no guarantee that the segments will be contiguous. In fact, the linker may place segments that do not belong to the group in the same 64K bytes of memory. Although **link** does not explicitly check whether all segments in a group fit within this 64K of memory, the linker is likely to encounter a fixup-overflow error if this requirement is not met.

Fixups

Once the starting address of each segment in a program is known, and all segment combinations and groups have been established, the linker can fix up any unresolved references to labels and variables. To fix up unresolved references, the linker computes an appropriate offset and segment address and replaces the temporary values, generated by the assembler, with the new values.

link carries out fixups for four different references:

- Short
- Near self-relative
- Near segment-relative
- Long

The size of the value to be computed depends on the type of reference. If **link** discovers an error in the anticipated size of a reference, it displays a fixup-overflow message. This error can occur, for example, if a program attempts, by using a 16-bit offset, to reach an instruction in a segment that has a different frame address. The error can also occur if the segments in a group do not fit within a single 64K-byte block of memory.

Short References

A *short reference* occurs in JMP instructions that attempt to pass control to labeled instructions in the same segment or group. The target instruction must be no more than 128 bytes from the point of reference. The linker computes a signed 8-bit number for this reference and displays an error message if the target instruction belongs to a different segment or group (that is, if it has a different frame address), or if the target is more than 128 bytes distant (in either direction).

Near Self-Relative References

A *near self-relative reference* occurs in instructions which access data relative to the same segment or group. The linker computes a 16-bit offset for this reference and displays an error message if the data are not in the same segment or group.

Near Segment-Relative References

A *near segment-relative reference* occurs in instructions that attempt to access data in a specified segment or group, or that is relative to a specified segment register. **link** computes a 16-bit offset for this reference and displays an error message if either of the following conditions exists; the offset of the target within the specified frame is greater than 64K bytes or less than 0, or the beginning of the canonical frame of the target is not addressable.

Long References

A *long reference* occurs in CALL instructions that attempt to access an instruction in another segment or group. **link** computes a 16-bit frame address and 16-bit offset for this reference and displays an error message if either of the following conditions exists; the computed offset is greater than 64K bytes or less than 0, or the beginning of the canonical frame of the target is not addressable.

Chapter 9

debug: A Debugger

Introduction

In this chapter you'll learn how to:

- start the **debug** utility,
- use the **debug** commands and parameters.

The **debug** utility is a debugging program that provides a controlled testing environment for binary and executable object files. Note that **edlin**, the MS-DOS line editor, is used to alter source files; **debug** is **edlin**'s counterpart for binary files.

debug eliminates the need to reassemble a program to see if a problem has been fixed by a minor change. It allows you to alter the contents of a file or the contents of a CPU register, and then immediately re-execute a program to check the validity of the changes made.

All **debug** commands can be aborted at any time by pressing <CTL>c. The <CTL>s key sequence suspends the display, so that you can read it before the output scrolls away. Pressing any key other than <CTL>c or <CTL>s restarts the display. All these commands are consistent with the control character functions available at the MS-DOS command level.

Starting debug

debug can be started in two ways. Using the first method, you type all commands in response to the **debug** prompt (a hyphen). Using the second method, you type all commands on the command line used to start **debug**.

To start **debug** using the first method, simply type the following:

debug

debug responds with the hyphen (–) prompt, signaling that it is ready to accept your commands. Since you didn't specify a filename, you can use other commands to work on current memory, disk sectors, or disk files. (The **debug** commands are described later in this chapter.)

To start **debug** using the second method, you must use the following syntax:

debug [*filename* [*arglist*]]

For example, to start **debug** specifying a filename, you would type the following:

debug file.exe

debug would then load *file.exe* into memory starting at 100 (hexadecimal) in the lowest available segment. The BX:CX registers are loaded with the number of bytes placed into memory.

If you do include a *filename*, you might also specify an *arglist*. An *arglist* is a list of filename parameters and switches that are to be passed to the program *filename*. When *filename* is loaded into memory, it is loaded as if it had been started with a command of the form:

debug *filename arglist*

Here, *filename* is the file to be debugged, and *arglist* is the rest of the command line used when **debug** calls and loads *filename* into memory.

Using debug Commands

Each **debug** command consists of a single letter followed by one or more parameters. Additionally, the control characters and special editing functions described in Chapter 5, "MS-DOS Editing Keys," apply to **debug** as well.

If a syntax error occurs in a **debug** command, **debug** reprints the command line and indicates the error with a caret (^) and the word "Error" as in the following example:

```
dcs:100 cs:110
  ^ Error
```

Note that when typing commands and parameters you may use any combination of uppercase and lowercase letters.

The **debug** commands are listed below. Following this list, the commands and their parameters are described in greater detail.

debug Command	Function
A [*address*]	Assemble
C *range address*	Compare
D [*range*]	Dump
E *address* [*list*]	Enter
F *range list*	Fill
G [=*address* [*address...*]]	Go
H *value value*	Hex
I *value*	Input
L [*address* [*drive:record record*]]	Load
M *range address*	Move
N *filename* [*filename*]	Name
O *value byte*	Output
P [=*address*] [*value*]	Proceed
Q	Quit
R [*register-name*]	Register
S *range list*	Search
T [=*address*] [*value*]	Trace
U [*range*]	Unassemble
W [*address* [*drive:record record*]]	Write

You must be aware of the following when working with **debug**:

- When **debug** (version 2.0) is started, it sets up a program header at offset 0 in the program work area. In previous versions of **debug**, you could overwrite this header. You can still overwrite the default header if you don't give a filename to **debug**. If you are debugging a *.com* or *.exe* file, however, do not tamper with the program header below address 5CH, or **debug** will terminate.

- Do not restart a program after the following message is displayed:

```
Program terminated normally
```

  You must reload the program with the **N** (name) and **L** (load) commands for it to run properly.

debug Command Parameters

All **debug** commands accept parameters, except the **Q** (quit) command. Parameters may be separated by delimiters (spaces or commas), but a delimiter is required only between two consecutive hexadecimal values. Thus, the following commands are equivalent:

```
dcs:100 110
d cs:100 110
d,cs:100,110
```

Parameter	Definition
drive	A one-digit hexadecimal value that indicates which drive a file will be loaded from or written to. The valid values are 0 to 3, where 0=A:, 1=B:, 2=C:, and 3=D:.
byte	A two-digit hexadecimal value placed in, or read from an address or register.
record	1-digit to 3-digit hexadecimal value that indicates the logical record number on the disk and the number of disk sectors to be written or loaded. Logical records correspond to sectors; however, since they represent the entire disk space, their numbering differs.
value	A hexadecimal value of up to four digits that specifies a port number or the number of times a command should repeat its functions.
address	A two-part designation containing either an alphabetic segment register or a four-digit segment address plus an offset value. You may omit the segment name or segment address, in which case the default segment DS is used for all commands except G, L, T, U, and W, for which the default segment is CS. All numeric values are hexadecimal.

The following is an example address:

```
CS:0100
04BA:0100
```

The colon is required between the segment name (whether numeric or alphabetic) and the offset value.

range Contains two addresses: for example, *address address*; or one address, an L, and a value: for example, *address* L *value* where *value* is the number of lines on which the command should operate (L80 is assumed). The second type of *range* cannot be used if another hexadecimal value follows, since the hexadecimal value would be interpreted as the second address of the *range*.

Here are some example *ranges*:

```
CS:100 110
CS:100 L 10
CS:100
```

The following example, however, is illegal:

```
CS:100 CS:110
^Error
```

The limit for *range* is 10000 (hexadecimal). To specify a value of 10000 with only four digits, type 0000 (or 0).

list A series of byte values or strings. *list* must be the last parameter on the command line.

Following is an example *list*:

```
fcs:100 42 45 52 54 41
```

string Any number of characters enclosed in quotation marks. The quotation marks may be either single (‘ ’) or double (" "). If the delimiter marks must appear within a string, you must use the double quotation marks.

For example, the following strings are legal:

```
"This 'string' is okay."
"This ""string"" is okay."
```

However, this string is illegal:

```
"This "string" is not okay."
```

Note that the double quotation marks are not necessary in the following *string*s:

```
"This ''string'' is not necessary."
```

The ASCII values of the characters in the *string* are used as a list of byte values.

assemble

The **assemble** command assembles 8086/8087/8088 mnemonics directly into memory.

Syntax

A[*address*]

Description

If a syntax error is found, **debug** responds with the following message, then redisplays the current assembly address:

```
^Error
```

All numeric values are hexadecimal and you must type them as 1 to 4 characters. Also, you must specify prefix mnemonics in front of the opcode to which they refer. You may type them on a separate line, however.

The segment override mnemonics are CS:, DS:, ES:, and SS:. The mnemonic for the far return is RETF. String manipulation mnemonics must explicitly state the string size. For example, use MOVSW to move word strings, and use MOVSB to move byte strings.

The assembler will automatically assemble *short*, *near*, or *far* jumps and calls, depending on byte displacement, to the destination address. You may override these jumps and calls by using a NEAR or FAR prefix, as in the following example:

```
0100:0500 JMP 502 ; a 2-byte short jump
0100:0502 JMP NEAR 505 ; a 3-byte near jump
0100:505 JMP FAR 50A ; a 5-byte far jump
```

You may abbreviate the NEAR prefix to NE, but the FAR prefix cannot be abbreviated.

debug cannot tell whether some operands refer to a word memory location or to a byte memory location. In this case, the data type must be explicitly stated with the prefix, WORD PTR or BYTE PTR. Acceptable abbreviations are WO and BY. For example:

```
NEG BYTE PTR [128]
DEC WO [SI]
```

debug also cannot tell whether an operand refers to a memory location or to an immediate operand. So, it uses the common convention that operands enclosed in square brackets refer to memory. For example:

```
MOV AX,21 ; Load AX with 21H
MOV AX,[21] ; Load AX with the
   ; contents
   ; of memory location 21H
```

Two popular pseudo-instructions are available with the **A** (assemble) command: the DB opcode, which assembles byte values directly into memory; and the DW opcode, which assembles word values directly into memory. Following are examples of both:

```
DB 1,2,3,4,"THIS IS AN EXAMPLE"
DB 'THIS IS A QUOTATION MARK: "'
DB "THIS IS A QUOTATION MARK: '"

DW 1000,2000,3000,"BACH"
```

The A command supports all forms of register indirect commands. For example:

```
ADD BX,34[BP+2].[SI-1]
POP [BP+DI]
PUSH [SI]
```

All opcode synonyms are also supported, as in the next example:

```
LOOPZ 100
LOOPE 100

JA 200
JNBE 200
```

For 8087 opcodes, the WAIT or FWAIT prefixes must be explicitly specified, as in these last examples:

```
FWAIT FADD ST,ST(3) ; This line assembles
    ; an FWAIT prefix
LD TBYTE PTR [BX] ; This line does not
```

compare

The **compare** command compares the portion of memory specified by range to a portion of the same size beginning at the specified address.

Syntax

C*range address*

Description

If the two areas of memory are identical, there is no display, and **debug** returns with the MS-DOS prompt. If there are differences, they are displayed in this format:

```
address1 byte1 byte2 address2
```

Example

The following commands have the same effect:

C100,1FF 300

or:

C100L100 300

Each command compares the block of memory from 100 to 1FFH with the block of memory from 300 to 3FFH.

dump

The **dump** command displays the contents of the specified range of memory.

Syntax

D[*range*]

Description

If you specify a range of addresses with the **D** (dump) command, the contents of the range are displayed. If you don't use parameters with the **D** command, 128 bytes are displayed at the first address (DS:100) after the address displayed by the previous **D** command.

The dump is displayed in two portions: a hexadecimal dump (each byte is shown in hexadecimal value) and an ASCII dump (the bytes are shown in ASCII characters). Non-printing characters are denoted by a period (.) in the ASCII portion of the display. Each display line shows 16 bytes, with a hyphen between the eighth and ninth bytes. Each displayed line begins on a 16-byte boundary.

Examples

If you type the command:

dcs:100 110

debug displays the dump in the following format:

```
04BA:0100 42 45 52 54 41 ... 4E 44 TOM SAWYER
```

If you simply type the **D** command, the display is formatted as described above. Each line of the display begins with an address, incremented by 16 from the address on the previous line.

Each subsequent **D** (typed without parameters) displays the bytes immediately following those last displayed.

If you type the following command, the display is formatted as described above, but 20H bytes are displayed:

DCS:100 L 20

If you then type the following command, the display is formatted as described above, but all the bytes in the range of lines from 100H to 115H in the CS segment are displayed:

DCS:100 115

enter

The **enter** command enters byte values into memory at the specified address.

Syntax

E*address*[*list*]

Description

If, when using the **E** (enter) command, you type the optional list of values, the byte values are replaced automatically. (If an error occurs, no byte values are changed.)

If you type the address without the optional list, **debug** displays the address and its contents, repeats the address on the next line, and then waits for your input. At this point, the **E** (enter) command waits for you to perform one of the following actions:

- Replace a byte value with a value you type. Simply type the value after the current value. If the one you type is not a legal hexadecimal value or if it contains more than two digits, the illegal or extra character is not echoed.

- Press <Space> to advance to the next byte. To change the value, simply type the new value as described in the previous action. If, when you press <Space>, you move beyond an 8-byte boundary, **debug** starts a new display line with the address displayed at the beginning.

- Type a hyphen (–) to return to the preceding byte. If you decide to change a byte behind the current position, typing the hyphen returns the current position to the previous byte. When you type the hyphen, a new line is started with its address and byte value displayed.

- Press <Return> to terminate the **E** command. <Return> may be pressed at any byte position.

Examples

Suppose you type the following command:

ECS:100

Now suppose that **debug** displays the following:

```
04BA:0100 EB._
```

To change this value to, say, 41, type the number 41 at the cursor, as shown:

```
04BA:0100 EB.41_
```

To step through the subsequent bytes, you would press <Space> until you saw the following:

```
04BA:0100 EB.41 10. 00. BC._
```

To change BC to the number 42, for instance, you would type the number at the cursor, as follows:

```
04BA:0100 EB.41 10. 00. BC.42_
```

Notice that the value 10 should be 6F. To correct this value, you would type the hyphen as many times as needed to return to byte 0101 (value 10), then replace 10 with 6F:

```
04BA:0100 EB.41 10. 00. BC.42-
04BA:0102 00.-
04BA:0101 10.6F_
```

Pressing <Return> ends the **E** command and returns you to the **debug** command level.

fill

The **fill** command fills the addresses in the specified range with the values in the specified list.

Syntax

F*range list*

Description

If the *range* contains more bytes than the number of values in the list, the list will be used repeatedly until all bytes in the range are filled.

If the *list* contains more values than the number of bytes in the *range*, the extra values in the *list* are ignored. If any of the memory in the *range* is not valid (bad or non-existent), an error will occur in all succeeding locations.

Example

Suppose you type the following command:

F04BA:100 L 100 42 45 52 54 41

debug would now fill memory locations 04BA:100 through 04BA:1FF with the bytes specified. The five values would then be repeated until all the 100H bytes were filled.

go

The **go** command executes the program currently in memory.

Syntax

G[=*address* [*address*...]]

Description

If you type the **G** (go) command by itself, the program currently in memory executes as if it had run outside **debug**.

If you set =*address*, execution of the **G** command begins at the address specified. The equal sign (=) is required so that **debug** can distinguish the start =*address* from the breakpoint *address*.

With the other optional *addresses* set, execution stops at the first address encountered, regardless of that address' position in the list of *addresses* that halt execution or program branching. When program execution reaches a breakpoint, the registers, flags, and decoded instruction are displayed for the last instruction executed. (The result is the same as if you had typed the **R** (register) command for the breakpoint address.)

You may set up to ten breakpoints, but only at addresses containing the first byte of an 8086 opcode. If you set more than ten breakpoints, **debug** returns the BP error message.

The user stack pointer must be valid and must have 6 bytes available for this command. The **G** command uses an IRET instruction to cause a jump to the program under test. The user stack pointer is set, and the user flags, Code Segment register, and Instruction Pointer are pushed on the user stack. (If the user stack is not valid or is too small, the operating system may crash.) An interrupt code (0CCH) is placed at the specified breakpoint address(es).

When **debug** encounters an instruction with the breakpoint code, it restores all breakpoint addresses to their original instructions. If you don't halt execution at one of the breakpoints, the interrupt codes are not replaced with the original instructions.

Example

Suppose you type the following command:

GCS:7550

The program currently in memory would execute up to the address 7550 in the CS segment. **debug** would then display registers and flags, after which the **G** command would terminate.

After **debug** has encountered a breakpoint, if you type the **G** command again the program runs as if you had typed the filename at the MS-DOS command level. The only difference is that program execution begins at the instruction after the breakpoint, rather than at the usual start address.

hex

The **hex** command performs hexadecimal arithmetic on the two specified parameters.

Syntax

H*value value*

Description

First, **debug** adds the two *values*, then subtracts the second *value* from the first. The results of these actions are displayed on one line – first the sum, then the difference.

Example

Suppose you type the following command:

H19F 10A

In response, **debug** would perform the calculations and then display the following result:

```
02A9 0095
```

input

The **input** command inputs and displays one byte from the port specified by value.

Syntax

I*value*

Description

A 16-bit port address is allowed.

Example

Suppose you type the following command:

```
I2F8
```

Suppose also that the byte at the port is 42H. **debug** would input the byte and then display the following:

```
42
```

load

The **load** command loads a file into memory.

Syntax

L[*address* [*drive*: *record record*]]

Description

Set BX:CX to the number of bytes read. The file must have been named either when you started **debug** or with the **N** (name) command. Both the **debug** invocation and the **N** command format a filename properly in the normal format of a File Control Block at CS:5C.

If you use the **L** (load) command without any parameters, **debug** loads the file into memory beginning at address CS:100 and sets BX:CX to the number of bytes loaded. If you type the **L** command with an address parameter, loading begins at the memory location specified by the address. If you use the **L** command with all parameters included, absolute disk sectors are loaded, instead of a file.

Each record is taken from the specified drive (the drive designation is numeric; 0=A:, 1=B:, 2=C:, etc.). **debug** begins loading with the first specified record, and continues until the number of sectors in the second record have been loaded.

Example

Suppose once you have started **debug** that you type the following commands:

```
-NFILE.COM
```

Now, to load *file.com*, you would simply type the **L** command.

debug would then load the file and display the **debug** prompt. Suppose now that you want to load only portions of a file or certain records from a disk. To do this, you would type the following:

```
L04BA:100 2 0F 6D
```

debug would then load 109 (6D hex) records, beginning with logical record number 15, into memory beginning at address 04BA:0100. Then, once it had loaded the records, **debug** would simply return the hyphen (-) prompt.

If the file has an *.exe* extension, it would be relocated to the load address specified in the header of the *.exe* file. The address parameter is always ignored for *.exe* files. The header itself is stripped off the *.exe* file before it is loaded into memory. So, the size of an *.exe* file on disk will differ from its size in memory.

If the file was named by the **N** (name) command, or specified when you started **debug**, as a *.hex* file, then typing the **L** command with no parameters would cause **debug** to load the file beginning at the address specified in the *.hex* file. If the **L** command included the option address, **debug** would add the address specified in the **L** command to the address found in the *.hex* file to determine the start address at which to load the file.

move

The **move** command moves the block of memory specified by *range* to the location beginning at the specified *address*.

Syntax

M*range address*

Description

Overlapping moves (i.e., moves where part of the block overlaps some of the current addresses) are always performed without loss of data. Addresses that could be overwritten are moved first. For moves from higher to lower addresses, the sequence of events is to first move the data beginning at the block's lowest address and then work toward the highest. For moves from lower to higher addresses, the sequence is to first move the data beginning at the block's highest address and then work toward the lowest.

Note that if the addresses in the block being moved will not have new data written to them, the data in the block before the move will remain. The **M** (move) command copies the data from one area into another, in the sequence described, and writes over the new addresses. This action is why the sequence of the move is important.

Example

Suppose you type the following command:

MCS:100 110 CS:500

In response, **debug** would first move address CS:110 to CS:510, then move CS:10F to CS:50F, and so on until CS:100 is moved to CS:500. To review the results of the move, you could type the **D** command, using the same address as you used with the **M** command.

name

The **name** command sets filenames.

Syntax

N*filename* [*filename...*]

Description

The **N** (name) command performs two functions. First, it assigns a filename for a later **L** (load) or **W** (write) command. So, if you start **debug** without naming a file to be debugged, you must type the command N*filename* before a file can be loaded. Second, the **N** command assigns filename parameters to the file being debugged. In this case, **N** accepts a list of parameters used by the file being debugged.

Note that these two functions overlap. Consider, for example, the following set of **debug** commands:

```
-NFILE1.EXE
-L
-G
```

The **N** command would use these commands to perform the following steps:

1. It would first assign the filename *file1.exe* to the file to be used in any later **L** or **W** commands.

2. It would also assign the *file1.exe* filename to the first filename parameter used by any program that is later debugged.

3. The **L** command would then load *file1.exe* into memory.

4. The **G** (go) command would cause *file1.exe* to be run with *file1.exe* as the single filename parameter (that is, *file1.exe* would be run as if *file1.exe* had been typed at the command level).

A more useful chain of commands might look like this:

```
-NFILE1.EXE
-L
-NFILE2.DAT FILE3.DAT
-G
```

In this example, the **N** command sets *file1.exe* as the filename for the subsequent **L** command, which loads *file1.exe* into memory. The **N** command is then used again, this time to specify the parameters to be used by *file1.exe*. Finally, when the **G** command is run, *file1.exe* is executed as if **file1 file2.dat file3.dat** had been typed at the MS-DOS command level.

Note that if you were to execute a **W** command now, then *file1.exe* – the file being debugged – would be saved with the name *file2.dat*. To avoid this kind of result, you should always execute an **N** command before either an **L** or **W** command.

There are four regions of memory that can be affected by the **N** command:

- CS:5CFCB for file 1
- CS:6CFCB for file 2
- CS:80 Count of characters
- S:81 All characters typed

The first *filename* parameter that you specify for the **N** command has a file control block (FCB) set up at CS:5C. If you name a second *filename* parameter, an FCB is set up for this parameter beginning at CS:6C. The number of characters typed in the **N** command (exclusive of the first character, N) is given at location CS:80.

The actual stream of characters given by the **N** command (again, exclusive of the letter N) begins at CS:81. Note that this stream of characters may contain switches and delimiters that would be legal in any command typed at the MS-DOS command level.

Example

A typical use of the **N** command is as follows:

```
DEBUG PROG.COM
-NPARAM1 PARAM2/C
-G
-
```

In this case, the **G** command executes the file in memory as if you had typed the following command line:

PROG PARAM1 PARAM2/C

Testing and debugging therefore reflect a normal runtime environment for *prog.com*.

output

The **output** command sends the specified byte to the output port specified by value.

Syntax

O*value byte*

Description

A 16-bit port address is allowed.

Example

Suppose you want to **debug** to output the byte value 4F to output port 2F8. To do this you could simply type the following command:

O2F8 4F

proceed

The **proceed** command executes one or more instructions and displays the contents of all registers, flags, and the decoded instructions.

Syntax

P[=*address*] [*value*]

Description

If you include the =*address* option in the **P** (proceed) command, execution begins at the specified address. If you include the segment designation in =*address* you must specify both CS and the IP. You can specify only the IP if you omit the segment designation from =*address*. The *value* option causes **debug** to execute the number of steps specified by *value*.

The **P** command executes any subroutine call, interrupt or looping machine instruction as one instruction. After proceed executes a subroutine call, **debug** returns to the user.

Example

Suppose you type the following command:

P

In response, **debug** would return a display of the registers, flags, and decoded instructions for that single instruction.

If you type the following command, **debug** executes sixteen (10 hex) instructions beginning at 011A in the current segment, and then displays all registers and flags for each instruction as it is executed. The display scrolls away until the last instruction is executed, and then stops. Now you can see the register and flag values for the last few instructions performed:

P=011A 10

Remember that if you want to study the registers and flags for any instruction (at any time), you can press <CTL>s to stop the display from scrolling.

quit

The **quit** command terminates the **debug** utility.

Syntax

Q

Description

The **Q** (quit) command takes no parameters and exits **debug** without saving the file you're currently working with. You are returned to the MS-DOS command level.

Example

To end the debugging session, simply type the following and press the <Return> key:

```
q
```

debug terminates, and control returns to the MS-DOS command level.

register

The **register** command displays the contents of one or more CPU registers.

Syntax

R[*register-name*]

Description

If you do not type a *register-name*, the **R** (register) command dumps the register storage area and displays the contents of all registers and flags.

If you do type a *register-name*, the 16-bit value of that register is displayed in hexadecimal, and a colon then appears as a prompt. You can now either type a value to change the register, or press the <Return> key if you don't want a change.

Following is a list of the valid *register-names*:

AX	BP	S
BX	SI	CS
CX	DI	IP
DX	DS	PC
SP	ES	F

(IP and PC both refer to the Instruction Pointer.)

Any other entry for *register-name* results in a BR error message.

If you type **F** as the *register-name*, **debug** displays each flag with a two-character alphabetic code. To change any flag, type the opposite two-letter code. The flags are then either set or cleared.

The flags are listed below with their codes for SET and CLEAR:

Flag name	Set	Clear
Overflow	OV	NV
Direction	DN Decrement	UP Increment
Interrupt	EI Enabled	DI Disabled
Sign	NG Negative	PL Plus
Zero	ZR	NZ
Auxiliary Carry	AC	NA
Parity	PE Even	PO Odd
Carry	CY	NC

Whenever you type the **RF** command, the flags are displayed (in the order shown in the previous table) in a row at the beginning of a line. At the end of the list of flags, **debug** displays a hyphen (–).

You may enter new flag values in any order as alphabetic pairs. You do not have to leave spaces between these values. To exit the **R** command, press the <Return> key. Any flags for which you did not specify new values remain unchanged.

If you type more than one value for a flag, **debug** returns a DF error message. If you enter a flag code other than one of those shown in the table above, **debug** returns a BF error message. In both cases, the flags up to the error in the list are changed; those flags at and after the error are not.

When you start **debug**, the segment registers are set to the bottom of free memory, the Instruction Pointer is set to 0100H, all flags are cleared, and the remaining registers are set to zero.

Examples

If you type the following command, **debug** displays all registers, flags, and the decoded instruction for the current location:

R

If the location is CS:11A, for example, the display will look similar to this:

```
AX=0E00 BX=00FF CX=0007 DX=01FF SP=039D BP=0000
SI=005C DI=0000 DS=04BA ES=04BA SS=04BA CS=04BA
IP=011A NV UP DI NG NZ AC PE NC
04BA:011A CD21 INT 21
```

If you then type the following command, **debug** will display these flags:

```
RF
NV UP DI NG NZ AC PE NC - _
```

Now, you could type any valid flag designation, in any order, with or without spaces. For example:

```
NV UP DI NG NZ AC PE NC - PLEICY
```

In response, **debug** would display the **debug** prompt. To see the changes, type either the **R** or **RF** command:

```
RF
NV UP EI PL NZ AC PE CY - _
```

Press the <Return> key to leave the flags this way or to specify different flag values.

search

The **search** command searches the specified range for the specified list of bytes.

Syntax

S*range list*

Description

The *list* may contain one or more bytes, each separated by a space or comma. If the *list* contains more than one byte, only the first address of the byte string is returned. If the *list* contains only one byte, all addresses of the byte in the *range* are displayed.

Example

Suppose you type the following command:

SCS:100 110 41

debug would display a response similar to this:

```
04BA:0104
04BA:010D
-type:
```

trace

The **trace** command executes one instruction and displays the contents of all registers, flags, and the decoded instruction.

Syntax

T[=*address*] [*value*]

Description

If you include the =*address* option in the **T** (trace) command, tracing occurs at the specified =*address*. The value option causes **debug** to execute and trace the number of steps specified by *value*.

The **T** command uses the hardware trace mode of the 8086 or 8088 microprocessor. Consequently, you may also trace instructions stored in ROM (Read Only Memory).

Example

Suppose you type the following command:

T

In response, **debug** would return a display of the registers, flags, and decoded instruction for that one instruction. Assuming, for this example, that the current position is 04BA:011A, **debug** might return the following display:

```
AX=0E00 BX=00FF CX=0007 DX=01FF SP=039D BP=0000
SI=005C DI=0000 DS=04BA ES=04BA SS=04BA CS=04BA
04BA:011A CD21 INT 21
```

If you type the following command, **debug** executes sixteen (10 hex) instructions beginning at 011A in the current segment, and then displays all registers and flags for each instruction as it is executed. The display scrolls away until the last instruction is executed, and then stops. Now you can see the register and flag values for the last few instructions performed:

T=011A 10

Remember that if you want to study the registers and flags for any instruction (at any time), you can press <CTL>s to stop the display from scrolling.

unassemble

The **unassemble** command disassembles bytes and displays the source statements that correspond to them, with addresses and byte values.

Syntax

U[*range*]

Description

The display of disassembled code looks like a listing for an assembled file. If you type the **U** (unassemble) command without parameters, 20 hexadecimal bytes are disassembled at the first address after that displayed by the previous **U** command. If you type the **U** command including the range parameter, then **debug** disassembles all bytes in range. But if you specify range only as an address, then 20H bytes are disassembled.

Example

Suppose you type the following command:

U04BA:100 L10

In response, **debug** would disassemble 16 bytes, beginning at address 04BA:0100:

```
04BA:0100 206472 AND [SI+72],AH
04BA:0103 69 DB 69
04BA:0104 7665 JBE 016B
04BA:0106 207370 AND [BP+DI+70],DH
04BA:0109 65 DB 65
04BA:010A 63 DB 63
0.sp 0.2
4BA:010B 69 DB 69
04BA:010C 66 DB 66
04BA:010D 69 DB 69
04BA:010E 63 DB 63
04BA:010F 61 DB 61
```

Now, suppose you type the following:

U04ba:0100 0108

The display would now show:

```
04BA:0100 206472 AND [SI+72],AH
04BA:0103 69 DB 69
04BA:0104 7665 JBE 016B
04BA:0106 207370 AND [BP+DI+70],DH
```

If the bytes in some addresses are altered, the disassembler alters the instruction statements. You can then type the **U** command for the changed locations, the new instructions viewed, and the disassembled code used to edit the source file.

write

The **write** command writes the file being debugged to a disk file.

Syntax

W[*address* [*drive*: *record record*]]

Description

If you do not use parameters with the **W** (write) command, BX:CX must already be set to the number of bytes to be written; the file is written beginning from CS:100. If you type the **W** command with just an address, then the file is written beginning at that address. If you have used a **G** (go) or **T** (trace) command, you must reset BX:CX before using the **W** command without parameters.

Note that if a file is loaded and modified, the name, length, and starting address are all set correctly to save the modified file (as long as the length has not changed).

You must have named the file either with the initial **debug** startup command or with the **N** (name) command (refer to the **N** command earlier in this chapter). Both the **debug** startup command and the **N** command properly format a filename in the normal format of a File Control Block at CS:5C.

If you include parameters when you use the **W** command, the write begins from the memory address specified; the file is written to the specified drive (the drive name is numeric; 0=A:, 1=B:, 2=C:, etc.). **debug** writes the file beginning at the logical record number specified by the first record. **debug** then continues to write the file until the number of sectors specified in the second record have been written.

Writing to absolute sectors is extremely risky because the process bypasses the file handler.

Examples

If you type the following command, **debug** will write the file to disk and then display the **debug** prompt:

```
W
```

Two examples are shown below.

```
W
-_

WCS:100 1 37 2B
```

debug writes out the contents of memory to the disk in drive B:, beginning with the address CS:100. The data written out starts in the disk logical record number 37H and consists of 2BH records. When the write is complete, **debug** displays the following prompt:

```
WCS:100 1 37 2B
-_
```

debug Error Messages

During a **debug** session, you may receive any of the following error messages. Each error ends the **debug** command under which it occurred, but does not end **debug** itself.

Error Code	Definition
BF	Bad flag. You attempted to change a flag, but the characters you typed were not one of the acceptable pairs of flag values. See the **R** (register) command for the list of acceptable flag entries.
BP	Too many breakpoints. You specified more than ten breakpoints as parameters to the **G** (go) command. Retype the **G** command using ten or fewer breakpoints.
BR	Bad register. While using the **R** command you typed an invalid register name. See the **R** command for the list of valid register names.
DF	Double flag. You typed two values for one flag. You may specify a flag value only once per **RF** command.

Appendix A

How to Configure Your System

Introduction

The configuration file *config.sys* is a file containing commands, which MS-DOS checks at startup. Each time you start MS-DOS, it searches the root directory of the drive from which it was booted for a file named *config.sys*.

The *config.sys* file allows you to configure your system with a minimum of effort. For example, you can add device drivers to your system by including special commands in your *config.sys* file.

If your MS-DOS disk does not have a *config.sys* file, you can use the MS-DOS line editor, **edlin**, to create one and then save it on the MS-DOS disk in your root directory. If *config.sys* already exists and you want to change it, you can use **edlin** to edit it.

config.sys Commands

The following commands are used in the *config.sys* file:

Command	Effect
break	Sets <CTL>c check.
buffers	Sets the number of sector buffers.
country	Allows for international time, date, and currency.
device	Installs the device driver in the system.
drivparm	Defines parameters for block devices.
fcbs	Specifies the number of FCBs that can be open concurrently.
files	Sets the number of open files that can access certain MS-DOS system calls.
lastdrive	Sets the maximum number of drives that you can access.
shell	Begins execution of the shell from a specific file (usually *command.com*).
stacks	Supports the dynamic use of data stacks.

These commands are described in detail in the following pages. A sample *config.sys* file is included at the end of this appendix.

break

The **break** command sets <CTL>**c** check.

Syntax

break=[on]

or

break=[off]

Description

Depending on the program you are running, you may use <CTL>**c** to stop an activity (for example, to stop sorting a file). Normally, MS-DOS checks to see whether you have pressed <CTL>**c** only while it is reading from the keyboard or writing to the screen or printer. Therefore, setting **break** on extends <CTL>**c** checking to other functions, such as disk reading and writing. The default setting is break=on.

Example

To turn off <CTL>**c** checking, put the following line in your *config.sys* file:

```
break=off
```

buffers

The **buffers** command allows you to set the number of disk buffers that MS-DOS allocates in memory at the time you start the system.

Syntax

buffers=*x*

Description

The *x* option is the number of disk buffers, from 2 to 255. A disk buffer is a block of memory that MS-DOS uses to hold data when reading or writing.

The default number of buffers is 2, but for applications such as word processors, a number between 10 and 20 provides the best performance. If you plan to create a lot of subdirectories, you may even want to increase the number of disk buffers to between 20 and 30. Remember, though, that buffers take up space in memory, so you should not increase their number to a value greater than 30.

Example

To create ten disk buffers, put the following line in your *config.sys* file:

```
buffers=10
```

country

The **country** command allows MS-DOS to use international time, date, currency, and case conversions.

Syntax

country=*xxx*[,[*yyy*][,[*drive*:]*filename*]]

Description

The *xxx* option is a number representing a specific country, and is set by the equipment manufacturer. Values for *xxx* range from 001 to 999. The following table shows the possible values for *xxx*:

Value	Country
001	United States
031	Belgium
033	France
034	Spain
039	Italy
041	Switzerland
044	United Kingdom
045	Denmark
046	Sweden
047	Norway
049	Germany
061	Australia
358	Finland
972	Israel

The *yyy* option indicates the code page to be used at system boot; there are two code pages per country.

The [*drive*:]*filename* option gives the name and location of the file that includes country specific information.

Example

The following example sets the country to France (=033) and converts international currency, time, date, and case to French conventions:

```
country=033
```

device

The **device** command installs the specified device driver on the system list.

Syntax

device=[*drive:*][*path*]*filename*[*argument*]

Description

argument includes any switches accepted by *filename*.

The standard installable device drivers provided with MS-DOS are *ansi.sys*, *driver.sys*, *display.sys*, *printer.sys*, and *ramdrive.sys*. For more information on these installable device drivers, refer to Appendix B, "Installable Device Drivers."

If you purchase a new device, like a mouse or a scanner, you generally will receive device driver software with that device. These installable device drivers can be installed using the *device* command. Once you have installed a device driver, make sure that the device driver is in the directory that you specify in any device commands.

Note

The device drivers *country.sys*, and *keyboard.sys* are loaded automatically by MS-DOS. Do not try to load either of these with the device command. If you do, it will "hang" your system (that is, MS-DOS will not start).

Example

If you plan to use the ANSI escape sequences described in Appendix B, you should create a *config.sys* file and include the following command:

```
device=ansi.sys
```

This command causes MS-DOS to replace all keyboard input and screen output support with the ANSI escape sequences.

drivparm

The **drivparm** command allows you to define parameters for block devices when you start MS-DOS, overriding the original MS-DOS device driver settings.

Syntax

drivparm=/d:*dd* [/c] [/f:*ff*] [/h:*hh*] [/n] [/s:*ss*] [/t:*ttt*]

Description

Setting **drivparm** overrides any previous block device driver definitions.

The *dd* parameter for the **/d:** switch specifies a logical drive number between 0 and 255. This means that drive number 0=A, 1=B, 2=C, etc.

The **/c** switch specifies that change-line (doorlock) support is required.

The *ff* option on the **/f:** switch specifies the form factor index, where:

0 = 320/360K bytes
1 = 1.2M bytes
2 = 720K bytes
3 = 8 single density
4 = 8 double density
5 = Hard disk
6 = Tape drive
7 = Other

The default values for the following parameters depend upon the form factor specified with the **/f:** switch. If you do not specify the **/f:** switch, **drivparm** uses a default of 720K bytes.

The *hh* option on the **/h:** switch specifies the maximum head number. Its value can range from 1 to 99.

The **/n** switch specifies a non-removable block device.

The *ss* option for the **/s:** switch specifies the number of sectors per track. Its value can range from 1 to 99.

The *ttt* option for the **/t:** switch specifies the number of tracks per side on the block device. Its value can range from 1 to 999.

Example

Suppose your computer has an internal tape drive on drive D: that is configured at startup to write 20 tracks of 40 sectors per track. If you want to reconfigure this tape drive to write 10 tracks of 99 sectors each, you can put the following line in your *config.sys* file:

```
drivparm=/d:3 /f:6 /h:1 /s:99 /t:10
```

This command line overrides the default device driver settings, and supports a tape drive as drive D: (in this case the logical and physical drive numbers are identical). This tape drive has 1 head and supports a tape format of 10 tracks and 99 sectors per track. (This assumes that the device driver for the tape device supports this configuration of tracks and sectors.) So, to create a tape that you can read on another computer, one which can read only this alternate format, you might want to use this method.

fcbs

The **fcbs** command allows you to determine the number of File Control Blocks (FCBs) that can be open concurrently.

Syntax

fcbs=*x*,*y*

Description

The *x* option represents the number of files that File Control Blocks can open at any one time. The default value for *x* is 4, but allowed values range from 1 to 255.

If an application tries to open more than *x* files by FCBs, the *y* option specifies the number of files opened by FCBs that MS-DOS cannot close automatically. The first *y* files opened by FCBs are protected from being closed. The default value is 0, but allowed values range from 1 to 255.

Example

To open up to four files by FCBs and to protect the first two files from being closed, put the following line in your *config.sys* file:

```
fcbs=4,2
```

files

The **files** command sets the number of open files that the MS-DOS system calls can access.

Syntax

files=*x*

Description

The *x* option represents the number of open files that the system calls can access. The default value for *x* is 8, although allowed values range from 8 to 255. The maximum number of files that one program can have open at a time is 20.

MS-DOS system calls 2FH through 60H are compatible with the UNIX operating system.

Example

To let MS-DOS open 20 files at one time, put the following line in your *config.sys* file:

```
files=20
```

lastdrive

The **lastdrive** command sets the maximum number of drives that you can access.

Syntax

lastdrive=x

Description

The *x* value represents the last valid drive that MS-DOS will accept. (*x* can be any letter from a to z, the default is e.) The minimum number of drives is equal to the number of drives that you have installed on your computer.

This command is useful only in a network environment. At start-up, MS-DOS recognizes five drives you have on your system. To make any extra drives valid, a network redirection must occur.

Note

MS-DOS allocates a data structure for each drive that you specify, so you should avoid specifying more drives than are necessary.

Example

The following command sets the last drive to *m*, unless you have added an external logical device with *driver.sys*. For information about *driver.sys*, see Appendix B, "Installable Device Drivers":

lastdrive=m

shell

The **shell** command begins execution of the shell (top-level command processor) from a file defined by the specified pathname.

Syntax

shell=[*drive*:]*pathname*

Description

The *pathname* option specifies the program that MS-DOS uses as a command processor. Instead of reading the standard *command.com*, MS-DOS starts the processor specified in *pathname*.

System programmers who write their own command processors (instead of using the MS-DOS file, *command.com*) should also use the **shell** command.

Example

The following command uses the file *\bin\newshell* as the command processor:

```
shell=\bin\newshell
```

stacks

The **stacks** command supports the dynamic use of stacks.

Syntax

stacks=n,s

Description

The *n* option represents the number of stacks, valid values for *n* range from 0 to 64. The *s* option represents the size of each stack, valid values for *s* range from 0 to 512.

When there is a hardware interrupt, MS-DOS allocates one stack from *n* stack specified. When **stacks**=0,0, MS-DOS will not switch stacks at interrupt time.

Default follows:

stacks	Computer
0,0	IBM-PC IBM-XT IBM PC-Portable
9,128	Other computers

Example

If you want to allocate 8 stacks of 512 bytes each for hardware interrupt handling, you would include the following command line in your *config.sys* file:

stacks=8,512

Sample config.sys File

A typical *config.sys* file might look like this:

```
buffers=10
files=10
device=\dev\network.sys
break=on
shell=a:\bin\command.com a:\bin /p
```

Note that the buffers and files commands are both set to 10.

To find the device that is being added to the system, MS-DOS searches for the pathname *\dev\network.sys*. This file is usually supplied on a disk with your device. Make sure that the device file is in the directory that you specify in the device command.

break is turned on in this example.

This file also sets the MS-DOS command processor to the *command.com* file located on the disk in drive A: in the *\bin* directory. **a:\bin** tells the command processor where to look for *command.com* if it needs to re-read the disk. The **/p** switch tells the command processor that it is the first program running on the system, so that it can process the MS-DOS **exit** command.

Appendix B

Installable Device Drivers

Introduction

ansi.sys, *driver.sys*, *display.sys*, *printer.sys*, and *ramdrive.sys* are five installable device drivers provided with MS-DOS. This appendix explains the details of these drivers. You should read Appendix A, "How to Configure Your System," for more information on how to install them.

The ansi.sys File

An ANSI escape sequence is a series of characters (beginning with an escape character or keystroke) that you can use to define functions for MS-DOS. Specifically, you can change graphics functions and affect the movement of the cursor. This appendix explains the ANSI escape sequences for MS-DOS.

To install *ansi.sys*, you must use the following syntax for the command line in your *config.sys* file:

device=[*drive*:] [*path*] ansi.sys

This section uses the following notation:

Pn represents the numeric parameter, a decimal number that you specify with ASCII digits.

Ps represents the selective parameter, a decimal number that you use to select a subfunction. You may specify more than one subfunction by separating the parameters with semicolons.

Pl represents the line parameter, a decimal number that you specify with ASCII digits.

Pc represents the column parameter, a decimal number that you specify with ASCII digits.

When a parameter is needed and none is specified, MS-DOS uses a default value.

Cursor Functions

The following escape sequences affect the position of the cursor on the screen.

CUP – Cursor Position

<ESC> [*Pl* ; *Pc* H

HVP – Horizontal & Vertical Position

<ESC> [*Pl* ; *Pc* F

CUP and HVP move the cursor to the position specified by the parameters. The first parameter specifies the line number; the second, the column number. The default value for *Pl* and *Pc* is 1. When no parameters are specified, the cursor moves to the home position (the upper left-hand corner of the screen).

CUU – Cursor Up

<ESC> [*Pn* A

This sequence moves the cursor up without changing columns. The value of *Pn* sets the number of lines moved. The default value is 1. If the cursor is already on the top line, MS-DOS ignores the CUU sequence.

CUD – Cursor Down

<ESC> [*Pn* B

This sequence moves the cursor down without changing columns. The value of *Pn* sets the number of lines moved. The default value is 1. If the cursor is already on the bottom line, MS-DOS ignores the CUD sequence.

CUF – Cursor Forward

<ESC> [*Pn* C

The CUF sequence moves the cursor forward without changing lines. The value of *Pn* sets the number of columns moved. The default value is 1. If the cursor is already in the far right column, MS-DOS ignores the CUF sequence.

CUB – Cursor Backward

<ESC> [*Pn* D

This escape sequence moves the cursor backwards without changing lines. The value of *Pn* sets the number of columns moved. The default value is 1. If the cursor is already in the far left column, MS-DOS ignores the CUB sequence.

DSR – Device Status Report

<ESC> [6 n

The console driver outputs a RCP (Restore Cursor Position) sequence when it receives the DSR escape sequence.

SCP – Save Cursor Position

<ESC> [s

The console driver saves the current cursor position. This position can be restored by the RCP sequence.

RCP - Restore Cursor Position

<ESC> [u

This sequence restores the cursor position to the value it had when the console driver received the SCP sequence.

Erase Functions

The following escape sequences affect erase functions.

ED – Erase Display

<ESC> [2 J

The ED sequence erases the screen. The cursor then goes to the home position.

EL – Erase Line

<ESC> [K

This sequence erases from the cursor to the end of the line (including the cursor position).

Screen Graphics

The following escape sequences affect screen graphics, but they work only if your monitor supports graphics functions. The parameters shown in italics are explained at the beginning of this appendix.

SGR – Set Graphics Rendition

<ESC> [*Ps* ; ... ; *Ps* m

The SGR escape sequence calls the graphics functions specified by the parameters described in the following list. These functions remain until the next occurrence of an SGR escape sequence.

Parameter	Function
0	All attributes off
1	Bold on
2	Faint on
3	Italic on
5	Blink on
6	Rapid blink on
7	Reverse video on
8	Concealed on
30	Black foreground
31	Red foreground
32	Green foreground
33	Yellow foreground
34	Blue foreground
35	Magenta foreground
36	Cyan foreground
37	White foreground
40	Black background
41	Red background
42	Green background
43	Yellow background

Parameter	Function
44	Blue background
45	Magenta background
46	Cyan background
47	White background
48	Subscript
49	Superscript

Parameters 30 through 47 meet the ISO 6429 standard.

SM – Set Mode

<ESC> [= *Ps* h <ESC> [= h <ESC> [= 0 h <ESC>[? 7 h

The SM escape sequence changes the screen width or type to one of the following:

Parameter	Function
0	40 × 25 black and white
1	40 × 25 color
2	80 × 25 black and white
3	80 × 25 color
4	320 × 200 color
5	320 × 200 black and white
6	640 × 200 black and white
7	Wraps at the end of each line

RM – Reset Mode

<ESC> [= *Ps* l <ESC> [= l <ESC> [= 0 l <ESC> [? 7 l

Parameters for RM are the same as for SM (Set Mode), except parameter 7 resets the mode that causes wrapping at the end of each line.

The driver.sys File

driver.sys is an installable device driver that supports external drives.

Note

If you want to configure a logical device, use the **drivparm** command described in Appendix A, "How to Configure Your System."

To install *driver.sys*, you must use the following syntax for the command line in your *config.sys* file:

device=driver.sys /d:*dd* **[/c] [/f:***ff***] [/h:***hh***] [/n] [/s:***ss***] [/t:***ttt***]**

The *dd* parameter on the **/d:** switch specifies a physical drive number between 0 and 255. Physical drives are numbered differently to logical drives. Floppy drives are numbered starting at 0, and hard drives are numbered starting at 80H.

For example, if you have a computer with two floppy drives, they might be numbered 0 and 1. If you were to add an external floppy drive, its physical drive number would be 02H.

If you have a computer with one floppy and one hard drive, the floppy drive is numbered 00H, but the hard drive is numbered 80H. If you add an external floppy drive, its physical drive number is still 02H because MS-DOS already knows the definitions of physical drives 00H and 01H.

The **/c** switch specifies that change-line (doorlock) support is required.

The *ff* option on the **/f:** switch specifies the form factor index, where:

0 = 320/360K bytes
1 = 1.2M bytes
2 = 720K bytes
3 = 8 single density
4 = 8 double density
5 = Hard disk
6 = Tape drive
7 = Other

The default values for the following parameters depend upon the form factor specified with the **/f:** switch. If you do not specify the **/f:** switch, *driver.sys* uses a default of 720K bytes.

The *hh* option on the **/h:** switch specifies the maximum head number. Its value can range from 1 to 99.

The **/n** switch specifies a non-removable block device.

The *ss* option on the **/s:** switch specifies the number of sectors per track. Its value can range from 1 to 99.

The *ttt* option on the **/t:** switch specifies the number of tracks per side on the block device. Its value can range from 1 to 999.

For example, if you want to add an external 720K byte drive to your computer, you would include the following line in the *config.sys* file:

```
device=driver.sys /d:02
```

The display.sys File

display.sys is an installable device driver that supports code page switching for the console device.

To install *display.sys*, insert a command line of the following form in your *config.sys* file:

device=[drive:] [*path*] **display.sys con[:]=[***type***[,***hwcp***]** [*,n,m*]]

The *type* option displays the adaptor in use. Valid values include MONO, CGA, EGA, and LCD.

The *hwcp* option is the code page supported by the hardware. The following values are allowed:

437 (United States)
850 (multilingual)
860 (Portugal)
863 (French-Canadian)
865 (Norway)

The *n* option represents the number of additional code pages that can be supported. This number is dependent on the hardware. MONO and CGA do not support other fonts, so *n* must be 0. EGA can be 2. LCD can be 1.

The *m* option represents the number of sub-fonts that are supported for each code page.

The printer.sys File

printer.sys is an installable device driver that supports code page switching for parallel ports LPT1, LPT2, and LPT3. (The port name PRN may be substituted for LPT1 to refer to the first parallel port.)

To install *printer.sys*, insert a command line of the following form in your *config.sys* file:

device=[drive:] [*path*] **printer.sys lptx=**[*type*[,*hwcp*[,...]] [,*n*]]

The *type* option indicates the printer in use.

The *hwcp* option represents the code page supported by the hardware. The following values are allowed:

437 (United States)
850 (Multilingual)
860 (Portugal)
863 (French-Canadian)
865 (Norway)

The *n* option is the number of additional code pages that can be supported. This number is dependent on the hardware. MONO and CGA do not support other fonts, so *n* must be 0. EGA can be 2. LCD can be 1.

The *m* option represents the number of sub-fonts that are supported for each code page.

The ramdrive.sys File

ramdrive.sys is a device driver that lets you use a portion of your computers memory as if it were a hard drive. This memory area is called a RAM disk and is sometimes referred to as a virtual disk.

RAM disks are much faster than disk drives because the information they contain is always loaded into memory. If your computer has extended memory installed (starting at the 1 megabyte boundary), or if you have an extended memory board that meets the Lotus®/Intel®/Microsoft Expanded Memory Specification, you can use this memory for one or more RAM disks. Otherwise, *ramdrive.sys* locates RAM drives in low memory.

Note

Any memory allocated to a RAM disk decreases the amount of memory available for MS-DOS applications and utilities.

To install *ramdrive.sys*, include a command of the following form in your *config.sys* file:

device=ramdrive.sys [*bbbb*] [*ssss*] [*dddd*] [/e]

or

device=ramdrive.sys [*bbbb*] [*ssss*] [*dddd*] **[/a]**

The *bbbb* option specifies the disk size in kilobytes. The default size is 64K bytes; the minimum, 16.

The *ssss* option specifies the sector size in bytes. The default value is 128 bytes. The following values are allowed; 128, 256, 512, and 1024 bytes.

The *dddd* option specifies the number of root directory entries. The default value is 64; the minimum 4; and the maximum 1024.

ramdrive.sys adjusts the value of *dddd* to the nearest sector boundary. For example, if you give a value of 25 when the sector size is 512 bytes, the 25 will be rounded up to 32, since 32 is the next multiple of 16 (there are sixteen 32 byte directory entries in 512 bytes).

The **/e** switch lets you use extended memory (above 1 megabyte) as a RAM disk if it has been installed. If you use this switch, you cannot use the **/a** switch. It is recommended that you use the **/e** switch.

The **/a** switch lets you use an extended memory board that meets the Lotus/Intel/Microsoft Expanded Memory Specification for a RAM drive, if that board has been installed. If you use this switch, you cannot use the **/e** switch.

Note

When you reset or turn off the power on your computer, any information stored in RAM disks is lost.

Appendix C

Disk and Device Errors

Introduction

This appendix describes MS-DOS disk and device error messages. See Appendix D, "MS-DOS Messages," for other MS-DOS messages.

If a disk or device error occurs at any time during a command or program, MS-DOS displays an error message in the following format:

type action device

Abort, Retry, Ignore:_

This appendix lists the *type*, *action*, and *device* messages in separate sections.

Type Messages

The *type* message is one of the following:

- **Bad call format error**
  The length of the request header passed to the device header was incorrect.

- **Bad command error**
  A device driver issued an incorrect command to the device specified in the error message.

- **Bad unit error**
  Invalid subunit numbers were passed to the device driver.

- **Data error**
  MS-DOS could not read the data from the disk properly. This is often due to a defective disk. Try choosing **R** (for Retry) several times, or choose **A** (for Abort) to end the program. (It's a good idea to make a new copy of the disk because, if it's defective, you may lose information.)

- **FCB unavailable**
  General failure error
  An unusual error has occurred. This error usually requires an experienced programmer to fix it. Choose **R** (for Retry) or **A** (for Abort).

- **Invalid disk change error**
  You changed the disk in a drive when you weren't supposed to. Put the disk back in the drive and choose **R** (for Retry).

- **Lock violation error**
  A program tried to access part of a file that another program was using. Choose **A** (for Abort), or wait a while and choose **R** (for Retry).

- **No paper error**
  The printer is either out of paper or not turned on.

- **Non-MS-DOS disk error**
  MS-DOS does not recognize the disk format because the disk is missing information or contains another operating system. Try running the **chkdsk** command to correct the problem. (See Appendix E, "MS-DOS Commands," for information about **chkdsk**.) If

running **chkdsk** does not solve the problem, you should reformat the disk by using the **format** command even though this will destroy all the files on the disk.

- **Not ready error**
  The device (usually a drive or printer) specified in the error message is not ready to accept or transmit data. This often happens when the disk drive door is open. If this is the problem, close the door and choose **R** (for Retry), or check to see if the printer is on and ready.

- **Read fault error**
  MS-DOS is unable to read data from the device (usually a disk drive). Check to see that the disk is properly inserted in the drive, then choose **R** (for Retry).

- **Sector not found error**
  This error usually means the disk has a defective spot so that MS-DOS cannot find the requested information on it. You should copy all files from the disk onto a good disk and then try to reformat the defective disk.

- **Seek error**
  MS-DOS is unable to locate the information on the disk. Make sure that the disk is properly inserted in the drive and choose R (for retry), or try a different drive.

- **Sharing violation**
  A program tried to access a file that another program was currently using. Choose **A** (for Abort), or wait a while and choose **R** (for Retry).

- **Write fault error**
  MS-DOS is unable to write data to the specified device. Make sure that the disk is properly inserted in the disk drive. Try **R** (for Retry). If the error occurs again, choose **A** (for Abort).

- **Write protect error**
  You tried to write data on a write-protected disk. If the disk has a write-protect tab on it, you must remove the tab before you can write on the disk. (You should consider first why the disk was write-protected.) If the disk doesn't have a write-protect notch, you cannot write on that disk.

Action Messages

The *action* message can be either of the following:

- reading
- writing

Device Messages

The *device* is the name of the device that has the error. Examples are device PRN (for a printer) or drive A: (for a disk drive). MS-DOS waits for you to type one of the following responses:

A **Abort**. End the program requesting the disk read or write.

I **Ignore**. Ignore the bad sector as though the error did not occur. This may result in lost data.

R **Retry**. Repeat the operation. You should use this response when you have corrected the error (for example, with "Not ready" or "Write protect" errors).

Usually, you will want to recover by typing responses in this order:

R (to try again)

A (to terminate a program and try a new disk)

One other error message might be related to faulty disk reading or writing:

```
File Allocation Table bad for drive x
```

This message means that the copy of one of the allocation tables in memory points to nonexistent blocks. Possibly the disk was incorrectly formatted or not formatted before use. If this error persists after you have formatted the disk, it is unusable.

Appendix D

MS-DOS Messages

Introduction

This appendix describes MS-DOS messages, their causes, and how to correct or respond to them. Following each error message, is the command(s) (in brackets) which caused the message.

For information on device error messages, see Appendix C, "Disk and Device Errors."

Messages

Abort edit (Y/N)?
[edlin]
MS-DOS displays this message when you choose the **edlin Q** (quit) command. The **Q** command exits the editing session without saving any editing changes. Type **Y** (for Yes) or **N** (for No).

Abort, Retry, Ignore?
[MS-DOS]
This is a device error. See Appendix C, "Disk and Device Errors."

Access denied
[attrib] [find] [print] [replace] [xcopy]
You tried to replace a write-protected, read-only, or locked file.

Add *filename*? **(Y/N)**
[replace]
replace displays this prompt if you specify the **/w** switch. Press **Y** if you want to add the file to the disk, or **N** if you do not want to add the file.

Adding *filename*
[replace]
replace displays this prompt to let you know that it is adding this file to your disk.

All files canceled by operator
[print]
MS-DOS displays this message when you specify the **/t** switch with the **print** command.

Allocation error, size adjusted
[chkdsk]
The size of the file indicated in the directory was not consistent with the amount of data actually allocated to the file. The file was truncated to match the amount of data allocated.

All specified file(s) are contiguous
[chkdsk]
All files are written sequentially on the disk. To correct this error automatically, specify the **chkdsk /f** switch.

APPEND/ASSIGN Conflict

[append]

You cannot use the **append** command on an assigned drive. Cancel the drive assignment before using the **append** command with this drive again.

Are you sure (Y,N)?

[MS-DOS]

MS-DOS displays this message if you try to delete all files in the working directory by using the ***.*** wildcard. Type **Y** (for Yes) to delete all the files, or **N** (for No).

Attempted write-protect violation

[format]

The disk you are trying to format is write-protected.

***** Backing up files to drive *x*: *****
Diskette Number: *n*

[backup]

backup displays this message while backing up files to the specified target drive. Be sure to label backup disks with the appropriate backup disk number for use in restoring them later.

Bad call format reading (or writing) drive *x*:

[MS-DOS]

This is a device error. See Appendix C, "Disk and Device Errors."

Bad command error reading (or writing) drive *x*:

[MS-DOS]

This is a device error. See Appendix C, "Disk and Device Errors."

Bad command or file name

[MS-DOS]

The command cannot find the program you asked it to run. Check to see that you typed the command line properly, and that the file or command is on the disk, or in the command path.

Bad or missing Command Interpreter

[MS-DOS]

MS-DOS cannot find the *command.com* file on the disk; either the file is missing from the root directory, or the file is invalid. You also receive this message if *command.com* has been moved from the directory it was originally in when you started MS-DOS. Either restart the system with a disk that contains the *command.com* file, or copy the *command.com* file from your backup MS-DOS master disk onto the disk used to start MS-DOS.

Bad or missing filename
[MS-DOS]
You specified a device incorrectly in the *config.sys* file. Check the accuracy of the device command in the *config.sys* file.

Bad Partition Table
[format]
This message means that there is no MS-DOS partition on the hard disk. You must run **fdisk** to create an MS-DOS partition on your hard disk.

Bad unit error reading drive *x*:
[MS-DOS]
This is a device error. See Appendix C, "Disk and Device Errors."

BREAK is off (or on)
[MS-DOS]
This message tells you the current setting of **break**.

Cannot CHDIR to path –
tree past this point not processed
[chkdsk]
chkdsk is checking the structure of the directory and is unable to go to the specified directory. All subdirectories underneath this directory will not be verified. To correct this error automatically, specify the **chkdsk /f** switch.

Cannot CHDIR to root
[chkdsk]
chkdsk is checking the tree structure of the directory and is unable to return to the root directory. **chkdsk** is not able to continue checking the remaining subdirectories. Try to restart MS-DOS. If this error persists, the disk is unusable.

Cannot CHKDSK a Network drive
[chkdsk]
You cannot check drives that are redirected over the network.

Cannot CHKDSK a SUBSTed or ASSIGNed drive
[chkdsk]
You cannot check drives that have been substituted or assigned.

Cannot COPY from (or to) a reserved device
[xcopy]
You cannot copy files from or to a device.

Cannot create Subdirectory BACKUP on drive x:
[backup]
The disk may be write-protected, full, or the backup subdirectory may already exist and be read-only. Use another disk as a target disk.

Cannot DISKCOMP to or from an ASSIGNed or SUBSTed drive
[diskcomp]
One of the drives that you specified is a drive that you created using the **assign** or **subst** command.

Cannot DISKCOMP to or from a network drive
[diskcomp]
You cannot compare disks on drives that have been redirected over the network.

Cannot DISKCOPY to or from an ASSIGNed or SUBSTed drive
[diskcopy]
One of the specified drives was created using the **assign** or **subst** command.

Cannot DISKCOPY to or from a network drive
[diskcopy]
You cannot copy disks to or from drives that have been redirected over the network.

Cannot do binary reads from a device
[copy]
The copy cannot be done in binary mode when you are copying from a device. You should either use the **/a** switch to specify an ASCII copy, or not use the **/b** switch.

Cannot edit .BAK file – rename file
[edlin]
You attempted to edit a file that had a filename extension of *.bak* (a backup copy created by **edlin**). If you must edit a file that has an extension of *.bak*, you must either rename the file or copy it and give it a different extension.

Cannot format an ASSIGNed or SUBSTed drive
[format]
You attempted to format a drive currently mapped to another drive by the **assign** or **subst** command. Run **assign** or **subst** again and clear all drive assignments.

Cannot FORMAT a Network drive
[format]
You cannot format drives that are redirected over the network.

Cannot JOIN a Network drive
[join]
You cannot join drives that are redirected over the network.

Cannot LABEL a Network drive
[label]
You cannot label a drive that is shared on a network server station.

Cannot LABEL a SUBSTed or ASSIGNed drive
[label]
You cannot label a drive if it has been substituted with the **subst** command or assigned with the **assign** command. Check the command line to be sure you specified a valid filename.

Cannot perform a cyclic copy
[xcopy]
When you are using the /s switch, you cannot specify a target that is a subdirectory of the source.

Cannot recover . entry, processing continued
[chkdsk]
The "." entry (working directory) is defective and cannot be recovered.

Cannot recover .. entry,
Entry has a bad attribute (or link or size)
[chkdsk]
The ".." entry (parent directory) is defective and cannot be recovered. If you have specified the **/f** switch, **chkdsk** tries to correct the error automatically.

Cannot RECOVER a Network drive
[recover]
You cannot recover files on drives that are redirected over the network.

Cannot SUBST a Network drive
[subst]
You cannot substitute drives that are redirected over the network.

Cannot SYS to a Network drive
[sys]
You cannot transfer the system files to drives that are redirected over the network.

Cannot use PRINT – Use NET PRINT
[print]
You must use the **net print** command to print files.

CHDIR .. failed, trying alternate method

[chkdsk]

When checking the tree structure, **chkdsk** was not able to return to a parent directory. It will try to return to that directory by starting over at the root and searching again.

Compare another diskette (Y/N)?

[diskcomp]

diskcomp displays this message when it has completed its comparison of the disks. Press **Y** (for Yes) if you want to compare more disks.

Compare error on disk side *s*, track *t*

[diskcomp]

diskcomp found a difference on the disk in the specified drive, side *s*, track *t*.

Compare OK

[diskcomp]

diskcomp displays this message if the disks are identical.

Compare process ended

[diskcomp]

diskcomp displays this message if a fatal error occurred during the comparison.

Comparing *t* tracks *n* sectors per track, *s* side(s)

[diskcomp]

This message confirms the format of the disks that you are comparing.

COM port does not exist

[mode]

You have specified an invalid COM port.

Contains *n* non-contiguous blocks.

[chkdsk]

The disk contains fragmented files. If you want to copy this disk, you should use the **copy** or **xcopy** command instead of the **diskcopy** command. The new copy will then store the new files sequentially.

Content of destination lost before copy

[copy]

The source file that you specified in the **copy** command was overwritten before the copy process completed. Refer to the **copy** command for the proper syntax.

Convert lost chains to files (Y/N)?

[**chkdsk**]

chkdsk displays this message if it finds information on the disk that is not allocated properly in the disk's File Allocation Table.

If you type **Y** (for Yes) in response to this prompt, **chkdsk** recovers the lost blocks it found when checking the disk. **chkdsk** then creates a proper directory entry and a file for each lost chain with a filename of the form *filennnn.chk*. If you type **N** (for No), **chkdsk** frees the lost blocks so that they can be reallocated and does not recover any data that was in those lost blocks.

Copy another diskette (Y/N)?

[**diskcopy**]

The **diskcopy** command has completed processing. Type **Y** (for Yes) if you want to copy another disk, or **N** (for No) if you do not.

Copying *t* **tracks**
n **Sectors/Track,** *s* **Sides**

[**diskcopy**]

diskcopy displays this message during copying.

Copy process ended

[**diskcopy**]

diskcopy could not copy the entire disk. Use the **copy** or **xcopy** command to copy specific files onto the disk.

Copyright 1981,82,83,84,85,86 Microsoft Corp.

[MS-DOS]

This message appears on most MS-DOS utility and command banners.

Corrections will not be written to disk

[**chkdsk**]

There are errors on the disk, but **chkdsk** will not correct them because you did not specify the **/f** switch. You must specify the **chkdsk** switch to correct disk errors.

Current date is *mm-dd-yy*

[**date**]

The **date** command displays this message. Enter the correct date and press <Return>.

Current time is *hh:mm:ss:cc*

[**time**]

The **time** command displays this message. Enter the correct time and press <Return>.

Data error reading drive *x*:
[MS-DOS]
This is a device error. See Appendix C, "Disk and Device Errors."

Delete current volume label (Y/N)?
[label]
If a current volume label exists, **label** displays this message in response to the prompt to enter the new volume label for this disk. If you want to delete the volume label, press **Y** (for Yes); otherwise, press **N** (for No).

DEVICE Support Not Present
[diskcomp] [diskcopy]
The disk drive does not support MS-DOS 4.0 device control.

Directory is joined
[chkdsk]
chkdsk does not process directories that are joined. Use the **join /d** command to "unjoin" the directories, and then run **chkdsk** again.

D

Directory is totally empty, no . or ..
[chkdsk]
The specified directory does not contain references to working and parent directories. Delete the specified directory and recreate it.

Directory not empty
[join]
You can only join onto an empty directory.

Disk error reading (or writing) drive *x*:
[MS-DOS]
This is a device error. See Appendix C, "Disk and Device Errors."

Disk error reading (or writing) FAT
[chkdsk]
One of your File Allocation Tables has a defective sector in it. MS-DOS automatically uses the other FAT.

You should copy all your files onto another disk. To correct this error automatically, simply specify the **chkdsk /f** switch.

Diskette bad or incompatible
[diskcopy]
The source disk is not formatted, or was formatted incorrectly. You cannot copy it.

Disk full. Edits lost

[edlin]

edlin was not able to save your file due to lack of disk space. You should always make sure that there is enough room on the default disk to save your file before you give the **edlin E** (end) command. You should also make sure that the default disk is not write-protected.

Disk unsuitable for system disk

[format]

The **format** program detected a bad track on the disk where system files should reside. You should use this disk to store data only.

Does name specify a file name or directory name on the target (F = file D = directory)?

[xcopy]

xcopy displays this prompt if the target directory does not exist. Type **F** if the name specifies a file, or **D** if the target specifies a directory that does not currently exist.

(.)(..) Does not exist

[chkdsk]

This is an informational message from **chkdsk**, indicating that either the "." or ".." directory entry is invalid.

Do not specify filename(s)
Command format: DISKCOMP d: d:[/1][/8]

[diskcomp]

You specified an incorrect switch or gave a filename in addition to a drive name.

Do not specify filename(s)
Command format: DISKCOPY d: d:[/1]

[diskcopy]

You specified an incorrect switch or gave a filename in addition to a drive name.

MS-DOS 2.0 or later required

[attrib] [backup] [fc] [graphics] [join] [mode] [restore] [subst]

You cannot use these utilities with 1.xx versions of MS-DOS.

Do you see the leftmost 0? (Y/N)

[mode]

mode displays this message to help you align the test pattern on your screen. Type **Y** (for Yes) if you can see the leftmost *0* in the test pattern, or type **N** (for No) if you want to shift the display to the right.

Do you see the rightmost 9? (Y/N)

[**mode**]

mode displays this message to help you align the test pattern on your screen. Type **Y** (for Yes) if you can see the rightmost *9* in the test pattern, or type **N** (for No) if you want to shift the display to the left.

Drive letter must be specified

[**format**]

You did not specify the drive letter for the drive that you want to format. You must specify the name of the drive that you want to format.

Drive *x*: not ready
Make sure a diskette is inserted into the drive and the door is closed

[**diskcomp**] [**diskcopy**]

The drive is empty, or you did not close the door of the disk drive.

Drive types or diskette types not compatible

[**diskcomp**] [**diskcopy**]

You must have the same size and type of disks to run these commands. For example, you cannot copy from a single-sided disk to a double-sided disk, or compare a high-density disk with a low-density disk. You should use **fc** if you want to compare the files on the disks. If you want to copy the disk, you can use **copy** or **xcopy**, or reformat the target disk so that it is the same type as the source disk, or use a disk of the same type.

Duplicate file name

[**rename**]

You tried to rename a file to a filename that already exists, or the name you specified could not be found.

ECHO is off (or on)

[MS-DOS]

This message tells you the current status of **echo**.

End of input file

[**edlin**]

The entire file was read into memory. If the file was read in sections, this message indicates that the last section of the file is in memory.

Enter current Volume Label for drive *x*:

[**format**]

format asks you to enter the current volume label for verification before it formats the hard disk in the specified drive. If you do not know what the volume label is, press <CTL>**c** to abort this command, and give the **vol** command for the specified drive. Then give the **format** command again.

Enter new date:

[date]

You must respond to this prompt when you start MS-DOS, or when you use the **date** command. Enter the date in a *mm-dd-yy* format, or press the <Return> key to accept the current date.

Enter new time:

[time]

You must respond to this prompt when you start MS-DOS. Enter the time in the *hh:mm* format, or press the <Return> key to accept the current time.

Entry error

[edlin]

The last command you typed contained a syntax error. Retype the command with the correct syntax and press the <Return> key.

Entry has a bad attribute (or link or size)

[chkdsk]

This message may be preceded by one or two periods that show which subdirectory is invalid. If you have specified the **/f** switch, **chkdsk** tries to correct the error automatically.

Error in .EXE file

[MS-DOS]

The *.exe* file you have asked MS-DOS to load has an invalid internal format. You cannot run this program. Check to make sure that you are using the correct version of MS-DOS.

Error reading/writing partition table

[format]

format could not read or write the partition table. You should run **fdisk** on the disk and then try formatting it again.

Errors found, F parameter not specified
Corrections will not be written to disk

[chkdsk]

chkdsk found errors on the disk. If you have not specified the **/f** switch, **chkdsk** continues printing messages but will not correct the errors. You should run **chkdsk** with the **/f** switch if you want to correct the problems encountered by the **chkdsk** command.

Errors on list device indicate that it may be off-line
Please check it

[print]

Your printer is not turned on.

Error trying to open backup log file
Continuing without making log entries.

[backup]
You specified the backup /L switch, but **backup** could not create the *backup.log* file.

Error writing to device

[MS-DOS]
You tried to send too much data to a device, so MS-DOS was unable to write the data to that device.

EXEC failure

[MS-DOS]
MS-DOS either found an error when reading a command, or the **files** command in the *config.sys* file is set too low. Increase the value of the **files** command in the *config.sys* file, and restart MS-DOS.

FCB unavailable reading (or writing) drive *x*:

[MS-DOS]
This is a device error. See Appendix C, "Disk and Device Errors."

fc: cannot open *filename* **– No such file or directory**

[fc]
One of the files that you specified does not exist. Check the directory for the correct filename.

fc: *filename* **longer than** *filename*

[fc]
After reaching the end of one of the files in a file comparison, the other file still has uncompared data left.

fc: incompatible switches

[fc]
You have specified switches that are not compatible. (For example, /b and /L.) You should not combine binary and ASCII comparison switches.

fc: no differences encountered

[fc]
The files are the same.

fc: out of memory

[fc]
You do not have enough memory to perform the comparison.

File allocation table bad

[MS-DOS]
The disk may be defective. Run **chkdsk /f** to check the disk.

File allocation table bad drive *x*:

[chkdsk]
This message means that the disk was not formatted or was formatted incorrectly. It could also mean that an operating system other than MS-DOS is on the disk. Run **chkdsk /f** to check the disk. If this message is displayed again, you must reformat the disk.

File *filename* canceled by operator

[print]
MS-DOS displays this message when you specify the **/t** switch with the **print** command.

File cannot be converted

[exe2bin]
The input file is not in the correct format.

File cannot be copied onto itself

[copy] [replace] [xcopy]
The source filename you specified is the same as the target filename.

File creation error

[MS-DOS] **[edlin] [restore] [xcopy]**
You tried to add a new filename or replace a file that already exists in the directory, or there was not enough space for the file. If the file already exists, it is a read-only file and cannot be replaced. This error message will also occur if the root directory is full or out of files, or if the filename is the same as a volume, a directory, or a hidden (or system) file.

File is READ-ONLY

[edlin]
The file is designated read-only, so you may not change it.

File name must be specified

[edlin]
You did not specify a filename when you started **edlin**. You should type the **edlin** command followed by a filename.

File not found

[chkdsk] [edlin] [fc] [find] [print] [recover] [rename] [xcopy]
MS-DOS could not find the file that you specified, or you tried to rename a file with a name already in the directory. Check to see that you entered the filename correctly.

File not in PRINT queue

[print]

The file that you specified was not in the print queue, so you cannot remove it from the queue. Check to see that you entered the filename correctly.

Files cannot be added to this diskette
Unless the PACK (/P) switch is used
Set the switch (Y/N)?

[backup]

The target disk does not have enough room for any of the files on the source disk without dividing them across disks. If you do not want to divide a file across disks, type **N** (for No). If your files are larger than will fit on one floppy disk, you must type **Y** (for Yes).

***** Files were backed up at** *time* **on** *date* *******

[restore]

This is an information message only.

FIND: Access denied

[find]

You cannot access the file. Make sure that the disk is not write-protected, read-only, or locked.

FIND: File not found

[find]

MS-DOS could not find the file that you specified. Make sure you have typed the filename correctly.

FIND: Invalid number of parameters

[find]

You specified either too many or too few options in the command line.

FIND: Invalid Parameter

[find]

One of the switches you specified is wrong.

FIND: Read error in filename

[find]

The **find** command could not read the specified file.

FIND: Syntax error

[find]

Check to make sure that you have typed the command correctly.

First cluster number is invalid, entry truncated
[**chkdsk**]
The file directory entry contains an invalid pointer to the data area. If you specified the **/f** switch, the file is truncated to a zero-length file.

FIRST diskette bad or incompatible
[**diskcomp**]
diskcomp cannot recognize the format on the source disk. You should run **chkdsk** to help you identify the problem.

Fixups needed – base segment hex:
[**exe2bin**]
The source (*.exe*) file contained information indicating that a load segment is required for the file. You must specify the absolute segment address where the finished module is to be located.

For cannot be nested
[MS-DOS]
You cannot nest **for** commands in a batch file.

Format another (Y/N)?
[**format**]
format displays this message when it has finished formatting a disk. Type **Y** (for Yes) if you want to format another disk, or type **N** (for No) if you do not. If you accidentally type **Y**, you can abort the format process by typing <CTL>**c** in response to the message "Strike any key."

Format complete
[**format**]
format displays this message when it has finished formatting the disk in the specified drive.

Format failure
[**format**]
MS-DOS could not format the disk. This message is usually displayed with an explanation of why the command failed.

Format not supported on drive *x*:
[**format**]
You cannot use **format** to format this drive. You may have specified device parameters that your computer does not support.

Formatting while copying
[**diskcopy**]
diskcopy displays this message if the target disk has never been formatted.

General failure reading (or writing) drive *x*:
[MS-DOS]
This is a device error. See Appendix C, "Disk and Device Errors."

Graftabl needs MS-DOS version 2.0 or later
[graftabl]
You cannot use **graftabl** with 1.xx versions of MS-DOS.

Graphics characters already loaded
[graftabl]
The **graftabl** command displays this message if you have already loaded the table of graphics characters into memory.

Graphics characters loaded
[graftabl]
The **graftabl** command displays this message after it loads the table of graphics characters into memory.

Has invalid cluster, file truncated
[chkdsk]
The file directory entry contains an invalid pointer to the data area. If you specified the **/f** switch, the file is truncated to a zero-length file.

Head: *hhh* **Cylinder:** *ccc*
[format]
format displays the head and cylinder number of the track currently being formatted.

Illegal device name
[mode]
Your computer does not recognize this device name.

Incompatible system size
[sys]
The system files occupy more space on the source disk than is available on the target disk. You cannot use the **sys** command to transfer the system files to this disk.

Incorrect MS-DOS version
[append] [attrib] [backup] [chkdsk] [diskcomp] [diskcopy] [edlin] [fc] [find] [format] [graphics] [join] [keyb] [label] [mode] [more] [print] [recover] [replace] [restore] [share] [sort] [subst] [sys] [tree] [xcopy]
Some MS-DOS utilities will not run on older versions of the operating system, and many are written to run only on the exact version of MS-DOS that they were created for. You must use the correct version of MS-DOS to run this command.

Incorrect number of parameters
[join] [subst]
You specified either too many or too few options in the command line.

Incorrect parameter
[assign] [share]
One of the options you specified is wrong.

Infinite retry on parallel printer timeout
[mode]
Your printer is probably off-line or not ready. If the printer appears to be ready, you may have to press the <CTL><ALT><DEL> keys simultaneously to reset the computer.

Insert backup diskette *n* into drive *x*:
[backup] [restore]
This message prompts you for the *n*th backup disk. Put the next disk into the specified drive. Be sure to label each backup disk in the appropriate order for use when restoring the files.

Insert destination disk in drive *x*: and strike any key when ready
[sys]
This message appears when you are using **sys** to transfer the operating system with a single disk drive. You should insert a disk in the appropriate drive and press any character or number key to begin processing.

Insert diskette for drive *x*: and strike any key when ready
[MS-DOS]
This message appears when MS-DOS is copying and formatting. You should insert a disk in the appropriate drive and press any character or number key to begin processing.

Insert diskette with batch file and press any key when ready
[MS-DOS]
The disk containing your batch file is not in the drive you originally specified. Re-insert the disk that contains the batch file in the appropriate drive.

Insert MS-DOS diskette in drive *x*: and strike ENTER when ready
[format]
You typed the **format** /s command, but the disk in the default drive does not contain MS-DOS system files. Insert a disk with the files *io.sys* and *msdos.sys* in the drive specified and press any key.

Insert FIRST diskette into drive *x*:
[**diskcomp**]
This message prompts you for the first disk that you want to compare.

Insert last backup diskette in drive *x*: Strike any key when ready
[**backup**]
This message prompts you for the final backup disk. After you have put the final backup disk into the drive specified, press any alphanumeric key to continue the backup process.

Insert restore target diskette into drive *x*:
[**restore**]
restore displays this prompt if you are restoring files to a floppy. Put the target disk into the specified drive.

Insert SECOND diskette into drive *x*:
[**diskcomp**]
This message prompts you for the disk that you want to compare with the first disk.

Insert source disk
[**backup**]
This message prompts you to put the source disk into the drive.

Insert SOURCE diskette into drive *x*:
[**diskcopy**]
This message prompts you to put the disk to be copied into the specified drive.

Insert system diskette in drive *x*: and strike any key when ready
[**sys**]
sys needs a disk from which to read the *io.sys* and *msdos.sys* files. Insert a system disk into the specified drive and press any character or number key to start the system copy process.

Insert TARGET diskette into drive *x*:
[**diskcopy**]
diskcopy displays this message to prompt you to place the target disk into the specified drive. If your computer has one floppy drive, this message prompts you to put the proper disk into the drive.

Insufficient disk space
[MS-DOS] [**replace**] [**sort**] [**xcopy**]
The disk is full and does not contain enough room to perform the specified operation.

Insufficient memory

[backup] [chkdsk] [diskcomp] [diskcopy] [edlin] [replace] [restore] [sort] [xcopy]

There is not enough memory in your computer to perform the specified operation. Before retrying this operation, you must free memory by deleting files. In **edlin**, you may be able to free memory by typing a **W** (write) command followed by an **A** (append) command.

Insufficient memory for system transfer

[format]

Your memory configuration is insufficient to transfer the MS-DOS system files *io.sys* and *msdos.sys* with the format /s switch.

Insufficient room in root directory. Erase files in root and repeat CHKDSK

[chkdsk]

chkdsk always recovers lost files into the root directory. In this case, your root directory is full. Delete some files in your root directory, or move them to another directory to make room for the lost files.

Intermediate file error during pipe

[MS-DOS]

The pipe operation uses temporary files on the disk that are deleted automatically once the piping process is complete. An error has occurred in one of these files. Make sure that there is enough room on the disk for the temporary file and that the disk is not write-protected, and try the command again.

Internal error

[fc] [mode] [share]

This message indicates an error in the utility.

Invalid argument

[backup] [fc] [restore]

You have specified an invalid argument. See Appendix E "MS-DOS Commands", for the correct syntax of the command, and try again.

Invalid baud rate specified

[mode]

You have specified an incorrect baud rate. Valid choices are 110, 150, 300, 600, 1200, 2400, 4800, 9600, and 19,200. You must specify at least the first two digits of the baud rate.

Invalid characters in volume label

[format] [label]

The volume label should only contain up to 11 alphanumeric characters.

Invalid COMMAND.COM
Insert COMMAND.COM disk in default drive and strike any key when ready

[MS-DOS]

The program you have just run used up almost all of your available memory. MS-DOS must now reload the *command.com* file from disk. However, either MS-DOS cannot find *command.com* on the disk, or the copy found is the incorrect version.

Insert a disk that contains a copy of *command.com* into the default drive (it must be the same version with which you started MS-DOS).

Invalid country code

[MS-DOS]

In your *config.sys* file you have specified a country number that is not in the table of files configured in this version of MS-DOS. Country codes must be in the range 1 to 99 and are set by your computer manufacturer.

Invalid current directory

[chkdsk]

Your disk has an invalid directory on it. You may be able to recover some of the files on this disk by copying them with the **copy** command. Otherwise, you must replace the disk.

Invalid date

[date] [xcopy]

You specified an invalid date in response to the date prompt. Enter a valid date. See Appendix E "MS-DOS Commands", for the correct syntax of the **date** command.

Invalid Date/Time

[backup]

You specified an invalid date with one of the **backup** command switches. See Appendix E "MS-DOS Commands", for the correct syntax of the **backup** command.

Invalid device

[MS-DOS]

The device specified was not AUX, CON, NUL, or PRN.

Invalid device parameters from device driver

[**format**]

format displays this message when the number of hidden sectors is not evenly divisible by the number of sectors per track (that is, the partition does not start on a track boundary). This might happen if you tried to format a hard disk that previously had been formatted with MS-DOS 2.x without first running **fdisk**, or if you have set the device driver parameters incorrectly. Check the *config.sys* file for incorrect **device** or **drivparm** commands.

Invalid directory

[MS-DOS]

The directory you specified either does not exist or is invalid. Check to see that you entered the directory name correctly.

Invalid disk change reading (or writing) drive *x*:

[MS-DOS]

This is a device error. See Appendix C, "Disk and Device Errors."

Invalid drive in search path

[MS-DOS]

The drive does not exist.

Invalid drive or filename

[**edlin**] [**recover**]

You did not type a valid drive name or filename. Enter a valid drive name or filename.

Invalid drive specification

[**backup**] [**chkdsk**] [**diskcomp**] [**diskcopy**] [**format**] [**label**] [**print**] [**replace**] [**restore**] [**sys**] [**tree**] [**xcopy**]

The drive is incorrect or does not exist. Enter a valid drive name.

Invalid environment size specified

[**command**]

You gave an invalid number of bytes with the **/e** switch. You must specify a number between 128 and 32768 (bytes).

Invalid number of parameters

[**attrib**] [**backup**] [**fc**] [**find**] [**recover**] [**restore**] [**xcopy**]

Either you did not specify an option or string, or you specified the wrong number of options in the command line.

Invalid parameter(s)

[backup] [chkdsk] [diskcomp] [diskcopy] [edlin] [find] [format] [join] [mode] [print] [replace] [restore] [sort] [subst] [sys] [tree] [xcopy]

One of the switches you specified is wrong or does not exist. See Appendix E "MS-DOS Commands", to make sure you are using the correct switches.

Invalid path, not directory, or directory not empty

[MS-DOS]

You are unable to remove the directory requested for one of the specified reasons.

Invalid path (or file not found)

[attrib] [backup] [copy] [restore] [tree] [xcopy]

You have entered a pathname or filename that does not exist. Enter a valid pathname or filename with the command.

Invalid sub-directory entry

[chkdsk]

The subdirectory that you specified either does not exist or is invalid. Check to see that you typed the subdirectory name correctly.

Invalid time

[time]

You specified an invalid time. See Appendix E "MS-DOS Commands", for the correct syntax, and try the command again.

Invalid Volume ID

[format]

format displays this message if you enter a volume label that does not match the label on the hard disk you want to format. It then quits the format process. Use the **vol** command to find out what the volume label for the hard disk is, then try the command again.

Label not found

[MS-DOS]

Your batch file contains a **goto** command to a nonexistent label.

Last backup diskette not inserted
Insert last backup diskette in drive *x*:
Strike any key when ready

[backup]

This message prompts you for the final backup disk. After you have put the final backup disk into the drive specified, press any alphanumeric key to continue the backup process.

***** Last file not backed up *****

[**backup**]

backup could not back up the last file on the disk. This message may occur if there is no more room on the target disk. It may also occur if there was an error in the source file, or on the target disk. You may have to back up this file separately to another disk.

Line too long

[**edlin**]

During an **edlin R** (replace) command, the string given as the replacement caused the line to expand beyond the limit of 253 characters. You must divide the long line into two lines and retry the **R** command.

List output is not assigned to a device

[**print**]

When you first type the **print** command, MS-DOS asks you what device you want to specify as a printer. This message appears if **print** is set up for a device that does not exist.

Lock violation reading (or writing) drive *x*:

[MS-DOS]

This is a device error. See Appendix C, "Disk and Device Errors."

***x* lost cluster(s) found in *y* chains**
Convert lost chains to files (Y/N)?

[**chkdsk**]

chkdsk displays this message if it finds information on the disk that is not allocated properly in the disk's File Allocation Table.

If you type **Y** in response to this prompt, **chkdsk** recovers the lost blocks it found when checking the disk. **chkdsk** then creates a proper directory entry and a file for each lost chain with the filename of the form: *filennnn.chk*. If you did not specify the **/f** switch, **chkdsk** displays: "x bytes would be freed." If you type **N**, **chkdsk** frees the lost blocks so that they can be reallocated and does not recover any data that was in those lost blocks. If you did not specify the **/f** switch, **chkdsk** does nothing.

LPT#: not redirected

[**mode**]

mode could not redirect the parallel printer port. Check to see whether you have specified the proper options.

LPT#: redirected to COM#:

[**mode**]

Output on the parallel printer port will now be sent to this asynchronous communications port.

LPT#: set for 80
[mode]
The parallel printer port has been set for 80 columns.

LPT#: set for 132
[mode]
The parallel printer port has been set for 132 columns.

Memory allocation error. Cannot load MS-DOS, system halted
[MS-DOS]
Restart MS-DOS. If this error persists, make a new copy of the MS-DOS disk from your backup copy of the system disk.

--More--
[more]
Press <Space> to view more of the file or directory.

MORE: Incorrect MS-DOS version
[more]
The **more** command does not run on MS-DOS versions before 2.0.

Must specify destination line number
[edlin]
You did not specify the destination line number for an **edlin C** (copy) or **M** (move) command. Retype the command with a destination line number.

Must specify ON or OFF
[MS-DOS]
The command requires either an ON or OFF argument.

Name of list device [PRN]:
[print]
This prompt appears the first time that **print** is run and the **/d** switch is not specified. You can specify the name of any valid device, which then becomes the print output device. If you press the <Return> key, MS-DOS uses the default list device PRN.

New file
[edlin]
edlin prints this message if it does not find a file with the name you specified. If you are creating a new file, ignore this message. If you do not intend to create a new file, check to see whether you have correctly typed the name of the file that you wish to edit.

No appended directories
[append]
You did not specify a path with the **append** command.

No COM: ports
[**mode**]
Your computer does not have an asynchronous communications (serial) port.

No files added (or replaced)
[**replace**]
The **replace** command did not add or replace any files.

No files found *filename*
[**replace**]
replace could not find matching source or target files.

No free file handles.
Cannot start COMMAND.COM, exiting
[MS-DOS]
Restart MS-DOS. If this message persists, increase the files command value in the *config.sys* file.

Non-MS-DOS disk error reading (or writing) drive *x*:
[MS-DOS]
This is a device error. See Appendix C, "Disk and Device Errors."

Non-system disk or disk error
Replace and strike any key when ready
[**format**] [**sys**]
Replace the disk with the proper disk and press any alphanumeric key to continue.

No paper error writing device dev
[MS-DOS]
This is a device error. See Appendix C, "Disk and Device Errors."

No path
[**path**]
You typed **path** and pressed the <Return> key to find out what your search path is, but you did not set a command search path.

No room for system on destination disk
[**sys**]
There is not enough room for the system files on the target disk. Delete some files to make room for the system files or use another disk. You may need to reformat the disk to put the system on it.

No room in directory for file
[**edlin**]
You tried to create or save a file to the root directory, but either it is full, or you specified an invalid disk drive or filename.

Check the command line that you used to start **edlin** for an invalid filename or disk drive entry. If your command contains no invalid entries, you should run the **chkdsk** program for the specified disk drive. If the status report shows that the disk directory is full, and if there is still enough memory left on the disk, you may be able to create the file in a subdirectory. (This is because subdirectories are not limited in size as is the root directory.) Otherwise, remove the disk and replace it with another formatted disk.

No room in root directory
[label]
There is not enough room in the root directory for a volume label. Delete or move some of the files from the root directory to make room for the volume label.

No space left on device
[backup] [fc] [restore]
You cannot back up or restore any more files, and you cannot send any more output from a file comparison to your disk because the target disk is now full. You should probably delete some of the files on the disk to make more room.

No sub-directories exist
[tree]
You have specified the /s switch, but the directory does not contain subdirectories.

No such file or directory
[backup] [fc] [restore]
One or more of the files or directories that you specified does not exist.

***** Not able to back up (or restore) file *****
[backup]
This message may occur if there was an error in the source file or on the target disk. Use the **chkdsk** command on the source disk to see if you can determine the problem.

Not a graphics printer file
[graphics]
The file you are printing does not contain graphics.

Not enough memory
[join] [share] [subst]
There is not enough memory for MS-DOS to run the command.

Not enough room to merge the entire file

[**edlin**]

There was not enough room in memory to hold the file during an **edlin T** (transfer) command. You must free some memory by writing some files to a disk or by deleting some files before transferring this file.

Not found

[**edlin**]

You specified an **edlin S** (search) or **R** (replace) command that was unable to find a further occurrence of the specified search or replace string.

Not ready error reading (or writing) drive *x*:

[MS-DOS]

This is a device error. See Appendix C, "Disk and Device Errors."

O.K.?

[**edlin**]

This prompt occurs during **edlin S** (search) or **R** (replace) command processing. If you press any key except **Y** (for Yes) or the <Return> key, the search or replace process continues.

Out of environment space

[**command**] [MS-DOS]

There is not enough room in the program environment to accept more data. To increase the size of the existing environment, use the **/e** switch with the **command** command or remove some of the existing environment variables by using the **set** command.

Parameters not compatible

[**format**] [**replace**]

You have specified switches that cannot be used together.

Parameters not compatible with fixed disk

[**format**]

You have used a switch that is not compatible with the specified drive.

Parameters not supported

[MS-DOS] [**format**]

You have specified parameters that MS-DOS does not support.

Parameters not supported by Drive

[**format**]

format displays this message when the device driver for this drive does not support Generic IOCTL function requests.

Path(name) too long
[print] [replace] [xcopy]
The pathname you specified was too long. You may have to change directories to use this command with files in deep subdirectories.

Path not found
[chkdsk] [replace] [subst] [xcopy]
You specified an invalid pathname.

Press any key to begin adding (replacing) file(s)
[replace]
When you specify the **/w** switch, **replace** displays this message to prompt you to start replacing files.

Press any key to begin formatting *x*:
[format]
This prompt is issued before you format a disk. Press any key to begin the format process or, if you wish to end this command, press <CTL>**c**.

Press any key to begin recovery of the *n* file(s) on drive *x*:
[recover]
This prompt is issued before you recover a disk or file. Press any key to begin the recovery. Recovered files are named *filennnn.rec*. If you wish to end this command, press <CTL>**c**.

Press any key when ready . . .
[diskcomp] [diskcopy]
This prompt gives you time to insert the appropriate disks before copying them. When you have inserted the disks into the appropriate drives, press any key to begin the **diskcopy** process. Or, if you wish to end this command, press <CTL>**c**.

Printer error
[mode]
The printer is off, or is not ready.

Printer lines per inch set
[mode]
mode has set the number of lines per inch for the printer.

PRINT queue is empty
[print]
There are no files waiting to be printed.

PRINT queue is full

[**print**]

There is only room for 10 files in the list of files waiting to be printed. You can make room for more by using the print /**q** switch. The limit is 32 files.

Probable non-DOS disk Continue (Y/N)?

[**chkdsk**]

The disk you are using is not recognized by this version of MS-DOS. The disk was either created by another system with a format that is not supported on this version of MS-DOS, or it is not an MS-DOS disk.

Do not continue processing if **chkdsk** returns this message for a floppy disk. If this message returns for a hard disk, the information describing the characteristics of the disk to MS-DOS has been destroyed. In this case, you may continue **chkdsk** processing by typing **Y** (for Yes). This error may mean that the File Allocation Table (FAT) is bad and that the disk is unusable.

Processing cannot continue

[**chkdsk**]

There is not enough memory in your machine to run **chkdsk** for this disk. You must obtain more memory to run **chkdsk**.

Program too big to fit in memory

[MS-DOS]

You need more memory to run your application. It is possible that some programs you have run are still using some memory. You can try to restart MS-DOS; however, if you still receive this message, you still need more memory.

Read error in filename

[**edlin**] [**find**]

MS-DOS could not read the entire file.

Read fault error reading drive *x*:

[MS-DOS]

This is a device error. See Appendix C, "Disk and Device Errors".

Reading source file(s)...

[**xcopy**]

xcopy is now reading the source files that you specified.

Reinsert diskette for drive *x*:

[**format**]

Re-insert the disk being formatted in the indicated drive.

Replace filename? (Y/N)

[replace]

replace displays this prompt if you specify the **/w** switch. Press **Y** if you want to replace the existing file, or **N** if you do not want to replace the file.

Replacing filename

[replace]

replace displays this prompt to let you know that it is replacing this file that exists on your disk.

Requested Screen Shift out of range

[mode]

You cannot shift the display any further.

Resident part of PRINT installed

[print]

This is the first message that MS-DOS displays when you issue the **print** command. It means that to process the **print** command with other processes, available memory has been reduced by several thousand bytes.

Resident portion of MODE loaded

[mode]

Part of the **mode** program is now resident in memory, and available memory has been reduced by several thousand bytes.

Restore file sequence error

[restore]

You have restored files in the wrong order. You must insert the backup disks in the same order that they were backed up.

***** Restoring files from drive *x*: *****
Diskette: *n*

[restore]

This message is displayed during the restore process.

Resynch failed. Files are too different

[fc]

FC compares what can be loaded into memory. If no lines match in the portion of the files in the buffer space, **FC** displays this message.

SECOND diskette bad or incompatible

[diskcomp]

The second disk does not contain the same format as the first disk, or **diskcomp** does not recognize the format of the second disk. You should run **chkdsk** to help you identify the problem.

Sector not found error reading (or writing) drive *x*:
[MS-DOS]
This is a device error. See Appendix C, "Disk and Device Errors."

Sector size too large in file *filename*
[MS-DOS]
The specified device driver loaded by *config.sys* uses a sector size larger than that of any other device driver on the system. You cannot run this device driver.

Seek error reading (or writing) drive *x*:
[MS-DOS]
This is a device error. See Appendix C, "Disk and Device Errors."

SHARE already installed
[**share**]
share can only be installed once.

Sharing violation reading drive *x*:
[MS-DOS]
This is a device error. See Appendix C, "Disk and Device Errors."

SORT: Incorrect MS-DOS version
[**sort**]
sort does not run on MS-DOS versions before 2.0.

SORT: Insufficient disk space
[**sort**]
The disk is full.

SORT: Insufficient memory
[**sort**]
There is not enough memory to run the **sort** program.

Source and target drives are the same
[**backup**] [**restore**]
You specified the same drive for the source and target disks.

Source disk is Non-removable
[**backup**]
This is an informational message indicating that the source disk is a hard disk.

Source does not contain backup files
[**restore**]
You are attempting to restore files from a disk that does not contain backup files.

Source is a floppy (or hard) disk
[**restore**]
This is an informational message only.

Source path required
[**replace**]
You did not specify a source path for the **replace** command.

Specified drive does not exist, or is Non-removable
[**diskcomp**] [**diskcopy**]
You cannot compare or copy hard disks with this command. You must specify the name of a valid floppy drive.

Specified MS-DOS search directory bad
[MS-DOS]
The **shell** command in the *config.sys* file is incorrect. Make sure that the *command.com* file exists and that MS-DOS can find it.

Strike a key when ready...
[MS-DOS]
This prompt occurs during command processing and is always accompanied by another message. This message is also displayed if you have inserted a **pause** command in a batch file. Usually, MS-DOS asks you to insert disks into appropriate drives before this prompt. To begin command processing, press any character, any number key, <Space>, or <Return>.

Syntax error
[**attrib**] [**find**] [MS-DOS]
You have entered a command incorrectly. Check to make sure you have typed the command correctly. Remember to enclose the **find** command string in double quotation marks.

System transferred
[**format**] [**sys**]
The system files were transferred during **format** or **sys** command processing.

Target cannot be used for backup
[**backup**]
Either the target disk has an unrecognizable format, or it is bad. Do not use the disk, or try to format the disk with the **format** command, or run **chkdsk** on it to determine the problem.

Target disk is Non-removable
[**backup**]
This is an informational message that the target disk is a hard disk.

Target diskette is write protected

[**diskcopy**]

The target disk either has a write-protect tab on it, or it does not have a write-protect notch. If you want to destroy any existing information on the disk, remove the write-protect tab and give the command again. If the disk does not have a write-protect notch, you cannot use it as a target disk.

Target diskette may be unusable

[**diskcopy**]

Either the target disk has an unrecognizable format, or it is bad. Try to format the disk with the **format** command, or run **chkdsk** on it to determine the problem.

Target is a floppy (or hard) disk

[**backup**]

This is an informational message only.

Target is full

[**restore**]

There is no more room on the target disk for restored files. You must delete some of the files on the disk to make room for these files, or use another disk.

Target is Non-Removable

[**restore**]

This is an informational message only.

Terminate batch job (Y/N)?

[MS-DOS]

If you press <CTL>**c** while in batch mode, MS-DOS asks you whether or not you wish to end batch processing. Press **Y** (for Yes) to end processing, or **N** (for No) to continue.

The last file was not restored

[**restore**]

There was not enough room on the target disk for the file, or the last file was bad. Use the **chkdsk** command to determine the problem.

Too many files open

[**edlin**] [**label**]

MS-DOS could not open the *.bak* file or write the volume label due to the lack of available system file handles. Increase the value of the **files** command in the *config.sys* file.

Too many open files

[**backup**] [**fc**] [**restore**] [**xcopy**]

MS-DOS could not open the files that you want to compare due to the lack of available system file handles. Increase the value of the **files** command in the *config.sys* file.

Track 0 bad – disk unusable

[**format**]

The **format** command can accommodate defective sectors on the disk, except for those near the beginning. You must use another disk.

Unable to create directory

[**mkdir**] [**xcopy**]

MS-DOS could not create the directory you specified. Check to see that there is not a name conflict. You may have a file with the same name, or the disk may be full.

Unable to erase

[**backup**]

backup could not erase the files on the target disk. Check to see that the files on the backup disk are not read-only, and that the disk is not write-protected.

Unable to shift Screen

[**mode**]

mode is unable to shift the test pattern on the screen any further.

Unexpected MS-DOS Error *n*

[**replace**]

An unexpected error *n* occurred, where *n* is the MS-DOS error number.

Unrecognized command in CONFIG.SYS

[MS-DOS]

There is an invalid command in your *config.sys* file. Refer to Appendix A, "How to Configure Your System," for a list of valid statements.

Unrecognized printer

[**graphics**]

You are using an invalid printer. Check to see whether you entered the command properly, or refer to Appendix E "MS-DOS Commands", to make sure that you have specified a valid printer name.

Unrecognized printer port
[graphics]
The printer device name that you specified was invalid. You may need to set the printer port by using the mode command.

Unrecoverable error in directory
Convert directory to file (Y/N)?
[chkdsk]
This message is displayed if **chkdsk** is unable to correct an error in a directory. If you respond **Y** (for Yes) to this prompt, **chkdsk** converts the bad directory into a file. You can then fix the directory or delete it. If you respond **N** (for No) to this prompt, you may not be able to write to or read from the bad directory.

Unrecoverable read (or write) error on drive *x*:
[MS-DOS]
This is a device error. See Appendix C, "Disk and Device Errors."

usage: fc [/a] [/b] [/c] [/l] [/lb n] [/w] [t] [/n] [/NNNN] file1 file2
[fc]
One of the switches that you have specified is invalid.

VERIFY is off (or on)
[MS-DOS]
This message tells you the current setting of the **verify** command.

Volume in drive *x*: has no label
[dir] [label] [vol]
This is an informational message displayed in response to the **dir**, **label**, or **vol** command.

Volume in drive *x*: is filename
[dir] [label] [vol]
This is an informational message displayed in response to the **dir**, **label**, or **vol** command.

Volume label (11 characters, ENTER for none)?
[format] [label]
This message is displayed when you specify the **label** command, or the /v switch in the **format** command. Type a volume label, or press <Return> to indicate that you do not want a volume label for this disk. It's a good idea, though, to specify a volume label to help you identify your disks.

WARNING, ALL DATA ON NON-REMOVABLE DISK DRIVE *x*: WILL BE LOST!
Proceed with Format (Y/N)?

[format]

This message appears when you try to format a hard disk that already contains data. If you press **Y** (for Yes) the data on the disk will be erased. If you do not want the files on your hard disk erased, press **N** (for No). Copy the files to a floppy disk and repeat the **format** command.

Warning!Directory full

[recover]

The root directory is too full for **recover** processing. Delete some files in the root directory to free space for the recovered files, and try the command again.

Warning! Diskette is out of sequence
Replace diskette or continue if okay
Strike any key when ready

[restore]

You should restore the diskettes in the order that you backed them up.

Warning! File *filename* is a hidden (or read-only) file
Replace the file (Y/N)?

[restore]

This message prompts you as to whether you want to replace a hidden or read-only file. Type **Y** (for Yes) if you want to restore the hidden or read-only file from the backup disk. Type **N** (for No) if you do not want to restore this file.

Warning! File *filename* was changed after it was backed up
Replace the file (Y/N)?

[restore]

This message prompts you as to whether you want to replace a backup file that has been changed. Type **Y** (for Yes) if you want to restore this file, or **N** (for No) if you do not.

Warning! Files in the target drive BACKUP (or root) directory will be erased

[backup]

backup found files in the target drive, and you did not specify the **/a** switch to append files.

Warning! No files were found to back up

[backup]

backup did not find any files to back up on the disk you specified.

Warning! No files were found to restore

[restore]

restore did not find the file that you wanted to restore from the backup disk.

Warning: Read error in EXE file

[exe2bin]

The amount read was less than the size of the header. This is a warning message only.

Write fault error writing drive *x*:

[MS-DOS]

This is a device error. See Appendix C, "Disk and Device Errors."

Write protect error writing drive *x*:

[MS-DOS]

This is a device error. See Appendix C, "Disk and Device Errors."

Appendix E

MS-DOS Commands

MS-DOS Commands

append	Sets a search path for data files.
assign	Assigns a drive letter to a different drive.
attrib	Sets or displays file attributes.
backup	Backs up one or more files from one disk to another.
break	Sets <CTL>**c** check.
chcp	Changes the code page.
chdir	Changes directories or prints the working directory (**cd**).
chkdsk	Scans the directory of the default or designated drive and checks for consistency.
cls	Clears the screen.
command	Processes internal MS-DOS commands.
comp	Compares the contents of two sets of files.
copy	Copies the specified file(s).
ctty	Changes device used for issuing commands.
date	Displays and sets the date.
del	Deletes the specified file(s) (**erase**).
dir	Lists the requested directory entries.
diskcomp	Compares disks.
diskcopy	Copies disks.
exe2bin	Converts executable (*.exe*) files to binary format.
exit	Exits the command processor and returns to the previous level.

fastopen	Reduces the time needed to open frequently-used new files.
fc	Comapres files and displays their differences.
fdisk	Configures hard disks for MS-DOS.
find	Searches for a constant string of text.
format	Formats a disk to receive MS-DOS files.
graftabl	Loads a table of graphics characters.
graphics	Prepares MS-DOS for printing graphics.
join	Joins a disk drive to a pathname.
keyb	Loads a keyboard program.
label	Labels disks.
mkdir	Makes a directory (**md**).
mode	Sets operation modes for devices.
more	Displays output one screen at a time.
nlsfunc	Used to switch on new country specific information.
path	Sets a command search path.
print	Prints files.
prompt	Allows you to change the prompt.
recover	Recovers a bad disk or file.
ren	Renames first file as second file (**rename**).
replace	Replaces previous versions of files.
restore	Restores backed up files.
rmdir	Removes a directory (**rd**).
select	Installs MS-DOS on a new floppy disk with desired country-specific information and keyboard layout.

set	Sets one string value to another in the environment, or displays the environment.
share	Installs file sharing and locking.
sort	Sorts data forward or backward.
subst	Substitutes a string for a pathname.
sys	Transfers MS-DOS system files from one drive to the drive specified.
time	Displays and sets the time.
tree	Displays directory and file names.
type	Displays the contents of a file.
ver	Prints the MS-DOS version number.
verify	Verifies all writes to a disk.
vol	Displays the volume label.
xcopy	Copies files and subdirectories.

These commands are described in detail in the following pages.

Introduction

This appendix lists the MS-DOS commands, in alphabetical order. For each command the syntax is given along with detailed descriptions, comments, and examples.

Some of the MS-DOS commands do not work over a computer network. If you try to use these commands, MS-DOS displays the error message "Cannot *command* to a network device," where *command* is the name of the command you typed.

The commands that do not work over a network are:

chkdsk	**fdisk**	**recover**	**subst**
diskcomp	**format**	**label**	**sys**
diskcopy			

Command Options

Command options give MS-DOS extra information about a command. If you do not include some options, MS-DOS provides values called *default values*. For the default values of particular commands, you should refer to individual command descriptions in this chapter.

MS-DOS commands use the following syntax:

command [*options*]

command is an MS-DOS command, and [*options*] is one or more of the following:

drive: Refers to a disk drive name. You need only specify a drive name if you are using a file that is not on the default drive.

filename Refers to any file and includes the filename extension (if there is one). The filename option does not refer to a device or drive name.

pathname Refers to a filename by the following syntax:

[*directory*][*directory* ...]*filename*

path Refers to a directory name by the following syntax:

[*directory*][*directory* ...]*directory*

switches Control MS-DOS commands. Switches begin with a slash, as in **/p**.

arguments Provide more information to MS-DOS commands. You usually choose between arguments; for example, *on* or *off*.

Conventions

The following conventions for command options are used:

- You must supply the text for any of the variable items shown in italics. For example, when *filename* is shown, you should type the name of your file.

- Items in brackets ([ ]) are optional. If you want to include optional information, you should only type the information within the brackets. Do not type the brackets.

- An ellipsis (...) means that you can repeat an item as many times as necessary.

- You must separate commands from their options by inserting certain characters or spaces. These characters are sometimes called *separators*. Generally, you should use spaces to separate commands from their options as in the following:

  rename *dull.doc sharpe.doc*

  You can also use a semicolon (;), an equal sign (=), or a tab to separate MS-DOS commands from their options.

  In this manual, spaces separate commands from their options.

- You should use punctuation marks, such as backslashes, slashes, equal signs, quotation marks, or colons, where appropriate.

- Disk drives are referred to as *source drives* and *target drives*. A source drive is the drive from which you transfer information. A target drive is the drive to which you transfer information.

- Many commands manipulate *strings* of text. A string is a group of characters that can include letters, numbers, spaces, and any other characters. Searching for a particular word in a file is a common use of a string.

append

sets a search path for data files

Syntax

First use only:

append [/x][/e]

To specify directories to be searched:

append [*drive*:]*path*[;[*drive*:][*path*]...]

To delete appended files:

append;

Description

The *append* command allows you to specify a search path for data files. The *path* option is the directory that MS-DOS searches for a data file. The *append* command will accept switches only the first time the command is invoked. *Append* accepts the following switches:

/x Extends the search path for data files. MS-DOS first searches the current directory for data files, if the required data files are not found, it searches the first directory in the *append* search path. If the files are still not found, MS-DOS continues to the second appended directory, and so on. MS-DOS will not search subsequent directories once the data files are located.

/e Causes appended directories to be stored in the MS-DOS environment.

You can specify more than one path to search by separating each with a semicolon (;). If you type the *append* command with the path option a second time, MS-DOS discards the old search path and uses the new one.

If you use *append* without options, MS-DOS displays the current data path. If you use the following command, MS-DOS sets a NUL data path:

append ;

This means that MS-DOS searches only the working directory for data files.

Notes

You can use the *append* command across a network to locate remote data files. If you are using the MS-DOS *assign* command, you must first use the *append* command. If you want to set a search path for external commands, see the *path* command in this chapter. *Append* searches the data path for all files regardless of their file extensions, only with the following MS-DOS system calls:

0FH	Open file (FCB)
23H	Get (FCB) file size
3DH	Open handle
11H	FCB search first (with /x switch only)
4EH	Handle find first (with /x switch only)
4BH	Exec (with /x switch only)

Examples

Suppose you want to access data files in a directory called *letters* on drive B, and in a directory called *reports* on drive A, to do this use the following command:

append b:\letters; a:\reports

Suppose you want to use the **/x** extension switch so that *append* first searches the current directory for data files before using the appended search paths. To do this, use the following before you type any other *append* command:

append /x

If you type the following command, MS-DOS first searches your current directory for data files. If the new data files are not in the current directory, MS-DOS searches the directory called *\neworder* on drive C. If the files are not in the *\neworder* directory, MS-DOS searches the *\bakorder* directory on drive C:

append c:\neworder; c:\bakorder

assign

assigns a drive letter to a different drive

Syntax

assign [*x*[=]*y*[...]]

Description

The *assign* command lets you read and write files on drives other than A and B for applications that use only those two drives.

In the above syntax line, *x* is the drive that MS-DOS currently reads and writes to, and *y* is the new drive that you want MS-DOS to read and write to.

Remember not to enter a colon after the drive letters *x* and *y*.

Examples

To reset all drives back to their original assignments, enter *assign*. Remember, though, you cannot assign a drive if it is being used by another program, and you cannot assign an undefined drive.

To ensure compatibility with future versions of MS-DOS, you should use *subst* instead of *assign*. The following commands, therefore, are equivalent:

```
assign a = c
```

is equivalent to

```
subst a: c:\
```

As a second example, the following command enables you to use drives other than A and B, such as a hard disk drive, C:

```
assign a=c b=c
```

All references to drives A and B would then go to drive C.

Notes

Because *assign* hides the true device type from commands that require actual drive information, you should not use this command with *backup* or *print* or during normal use of MS-DOS. Also note that two other commands, *format* and *diskcopy*, ignore drive reassignments.

assign is an external command.

attrib

sets or resets the read-only attribute of a file, or displays the attributes of a file

Syntax

attrib [+r|-r][+a|-a][*drive:*]*pathname* [/S]

Description

If an application opens a file with read and write permission, *attrib* forces read-only mode to allow file sharing over a network.

+r sets the read-only attribute of a file.

-r disables read-only mode.

+a sets the archive attribute of a file.

-a clears the archive attribute of a file.

[*drive:*]*pathname*
specifies the name of the drive and the full pathname of the file you want to reference.

/S makes *attrib* search through all files and all subdirectories.

The *backup*, *restore*, and *xcopy* (*mcopy*) commands use the archive attribute to control a selective *backup*/*restore*/*xcopy* on files that have been modified. You can use the *+a* and *-a* options to select files that you want to back up with the *backup /m* switch, or copy with the *xcopy* (or *mcopy*) */m* or */a* switches.

To display the attribute of a specific file on the default drive, enter *attrib* followed by the pathname of the file. To print the attributes of all files in a specific directory, use the wildcard abbreviation *.* in the pathname.

Examples

The following example gives the file *report.txt* read-only permission:

attrib +r report.txt

Suppose you want to use *xcopy* to copy all the files in the default directory of drive A, except for those that have an extension of *.bak*, to drive B. You would enter the following:

```
attrib +a a:*.*
attrib -a a:*.bak
```

and:

```
xcopy a: b: /m
```

or:

```
copy a: b: /a
```

If you use the */m* switch, *xcopy* automatically turns off the archive bits of the files in drive A as it copies them.

Notes

attrib is an external command.

backup

backs up one or more files from one disk to another

Syntax

backup [*drive*:][*path*][*filename*] [*drive*:][/**s**][/**m**][/**a**][/**d:***date*][/**t:***time*] [/**L:**[[*drive*:][*path*]*filename*]]

Description

The *backup* command can back up files on disks of different media. For example, you can back up files from:

- Hard disk to floppy disk
- Floppy disk to floppy disk
- Floppy disk to hard disk
- Hard disk to hard disk

The *backup* command also backs up files from one floppy disk to another even if the disks have a different number of sides or sectors.

The first *drive:* option is the name of the disk drive that you want to back up. The second *drive:* option is the name of the backup disk drive. You should note that unless you specify the */a* option, *backup* erases the old files on a backup disk before adding new files to it.

This *backup* program and the one supplied by IBM are compatible. File and disk formats are the same as those that the IBM *backup* program uses unless you set the */L* switch, which is described in the following list.

You can specify the following switches with *backup*:

/s Backs up subdirectories also.

/m Backs up only those files that have changed since the last backup.

/a Adds the files to be backed up to those already on the backup disk. It does not erase old files on the backup disk.

/d:date Backs up only those files that you last modified on or after the date specified.

/t:time Backs up only those files which have been modified at or after the specified time on the date specified with the */d:date* option.

/L Makes a backup log entry in the specified file. If you do not specify a filename, *backup* places a file called *backup.log* in the root directory of the disk that contains the files being backed up. The first line of the entry in the file contains [*date time*] where *date* and *time* are the backup dates and times. Each subsequent line in the entry corresponds to a backed-up file. These lines each consist of a filename and the number of the disk that contains the file.

/f Causes the target diskette to be formatted if it is not already formatted. This option works by executing the MS-DOS *format* command, which must be in the current path.

You can use this information when you need to restore a particular file from a floppy disk, but you must specify which disk to restore so that *backup* does not have to search for files. If the *backup.log* file already exists, *backup* appends the current entry to the file.

The *backup* command displays the name of each file as it is backed up. You should label and number each backup disk consecutively to help you restore the files properly with the *restore* command.

If you are sharing files, you can back up only the files that you have access to.

Exit codes

The *backup* command returns the following exit codes:

0 Normal completion

1 No files were found to back up

2 Some files not backed up due to sharing conflicts

3 Terminated by user

4 Terminated due to error

You can use the batch processing *if* command for error processing that is based on the *errorlevel* returned by *backup*.

Example

Suppose Emily wants to backup all of the files in the **\user\emily** directory on drive C to a blank, formatted disk in drive A. To do this, she would enter the following:

backup c:\user\emily a:

Notes

You should not use *backup* if the drive you are backing up has been assigned, joined, or substituted with *assign*, *join*, or *subst*. If you do, you may not be able to restore the files with *restore*.

backup is an external command.

break

sets <CTL>**c** check

Syntax

break [on]

or

break [off]

Description

Depending on the program you are running, you may use <CTL>**c** to stop an activity (for example, to stop sorting a file). Normally, MS-DOS checks to see whether you press <CTL>**c** while it is reading from the keyboard or writing to the screen or printer. If you set *break* to *on*, you extend <CTL>**c** checking to other functions such as disk reads and writes.

Examples

To check for <CTL>**c** only during screen, keyboard, and printer reads and writes, enter the following:

break off

To find out how *break* is currently set, enter the following:

break

Notes

break is an internal command.

Some programs may set themselves to respond to <CTL>**c** at any time. Setting *break* does not affect these programs.

chcp

change code page

Syntax

chcp [*nnn*]

Description

A code page lets the computer translate stored numeric values into characters that will either be displayed on a screen, printed on paper or entered from a keyboard. In effect, a code page contains the definition of a given character set. Code page switching can be used to choose between different character sets for use on devices such as printers or screens.

Several different code pages, which include character sets specific to different countries, are now supported. At system initialization, the code page supported is the default code page defined by the *country* command in the *config.sys* file.

nnn is the 3 digit number of the required code page.

The *chcp* command can be compared to the *mode* command. The difference is that *chcp* sets a new code page for all the devices that can support it, whereas *mode* switches the code page for individual devices. The *chcp* command cannot select a code page for a device unless that code page has been prepared for the device. See the manual entry for the *mode* command.

In order to use *chcp*, *nlsfunc* support must be loaded before the *chcp* command is issued. See the manual entry for *nlsfunc*.

The *chcp* program looks for the country information in the file specified by the *country* command in the system configuration file. If no *country* command is used then *chcp* looks at the current drive and directory for the file named *country.sys*. If the file cannot be found, a message is displayed to indicate this.

chdir, cd

changes to a different directory; displays the working directory

Syntax

chdir [*path*]

Description

If your working directory is *\user\emily* and you want to change to another directory (such as *\user\pete*), enter the following command:

chdir \user\pete

MS-DOS puts you in the new directory. There is also a shorthand notation for the *chdir* command:

cd ..

This command always puts you in the parent directory of your working directory.

Examples

If you use *chdir* without a path, you can display the name of your working directory. For example, if your working directory is *\user\pete* on drive B, and you enter *chdir*, MS-DOS displays the following:

b:\user\pete

The following command displays the name of the working directory on drive B:

chdir b:

Notes

chdir/cd is an internal command.

chkdsk

checks the disk in the specified drive

Syntax

chkdsk [*drive*:][*pathname*][/**f**][/**v**]

Description

You should run *chkdsk* occasionally on each disk to check for errors. If you do run *chkdsk* on a disk and any errors are found, *chkdsk* displays the error messages, followed by a status report.

A typical status report might look like this:

```
160256 bytes total disk space
8192 bytes in 2 hidden files
512 bytes in 2 directories
30720 bytes in 8 user files
121344 bytes available on disk

65536 bytes total memory
53152 bytes free
```

The chkdsk switches

/f Fixes errors on the disk. If you do not specify this switch, *chkdsk* does not correct errors that it finds in your directory, though it does display messages about files that need to be fixed.

/v *chkdsk* displays messages while it is running.

If you enter a filename after *chkdsk*, MS-DOS displays a status report for the disk and for the individual file.

Examples

If you want to save the *chkdsk* status report for future use, you can redirect the output from *chkdsk* to a file by entering the following:

```
chkdsk a:>filename
```

The errors are then sent to the specified file in this instance *filename*. Remember, though, not to use the */f* switch when you redirect *chkdsk* output.

If *chkdsk* finds errors on the disk in drive A and you want to try to correct them, enter the following command:

chkdsk a: /f

chkdsk now tries to correct any errors it encounters on the disk in drive A.

Notes

chkdsk does not correct errors on a disk unless you specify the */f* switch. For more information on *chkdsk* errors, refer to the specific error message in Appendix D, "MS-DOS Messages".

chkdsk is an external command.

cls

clears the screen

Syntax

cls

Description

This command clears your terminal screen by sending the ANSI escape sequence to your console.

Example

To clear your screen, enter the following:

```
cls
```

Notes

cls is an internal command.

command

starts the command processor

Syntax

command [*drive*:][*path*][*ctty-dev*] [**/e:***nnnnn*][**/p**][**/c** *string*]

Description

This command starts a new command processor (the MS-DOS program that contains all internal commands).

When you start a new command processor you also create a new command environment. This new environment is a copy of the old, parent, environment. However, you can change the new environment without affecting the old one.

The command processor is loaded into memory in two parts: transient and resident. Some applications write over the transient memory part of *command.com* when they run. When this happens, the resident part of the command processor looks for the *command.com* file on disk so that it can reload the transient part.

The *drive*: and *path* options tell the command processor where to look for the *command.com* file if it needs to reload the transient part into memory.

ctty-dev allows you to specify a different device (such as AUX) for input and output. See *ctty* in this appendix for more information.

/e:nnnnn
Specifies the environment size, where *nnnnn* is the size in bytes. The size may range between 128 and 32768 bytes. The default value is 128 bytes.

If *nnnnn* is less than 128 bytes, MS-DOS defaults to 128 bytes and displays the following message:

Invalid environment size specified

If *nnnnn* is greater than 32768 bytes, MS-DOS displays the same message, but defaults to 32768 bytes.

/p Tells *command.com* not to exit to any higher level shell.

/c string
Tells the command processor to perform the command or commands specified by *string* and then to return. If used, it should be the last switch in the command.

Example

The following command tells the MS-DOS command processor to do three things:

- Start a new command processor under the current program
- Run the command *chkdsk b:*
- Return to the first command processor:

command /c chkdsk b:

To learn how to use a pathname and the */p* switch with command, see the sample *config.sys* file in Appendix A, "How to Configure Your System".

Notes

command is an external command.

comp

compares the contents of two sets of files

Syntax

comp [*drive*:][*pathname*] [*drive*:][*pathname*]

Description

The *comp* command compares one file or set of files with a second file or set of files. These files can be on the same drive or on different drives. They can also be in the same directories.

The two sets of files you want to compare can have the same path and filenames provided they are on different drives. If you type only a drive for the second option, *comp* assumes that the second pathname is the same as the first. You can use wildcards (* and ?) to specify the pathnames.

If you do not type the pathname options, or if you omit the second pathname option *comp* prompts you for them. If either option contains only a drive or a path with no filename, *comp* assumes the filename is *.*.

If the files that you want to compare are on a different disk to *comp*, type the *comp* command with no options. When *comp* prompts you for the pathname options you can insert the correct disk and type the filenames to be compared.

As *comp* proceeds, it displays the paths and names of the compared files. A message appears if *comp* cannot find a file matching the second pathname, or if a directory path is invalid. If no file matches the first pathname option, *comp* prompts you for both the pathname options again.

During the comparison a message appears for any location in the two files that contain mismatching information. The message indicates the offset into the files of the mismatching bytes and the contents of the bytes themselves (all in hexadecimal notation). The message has the following format:

```
Compare error at OFFSET XXXXXXXX
file 1 = XX
file 2 = XX
```

In this format, *file1* is the first filename typed; *file2* is the second filename typed. After ten unequal comparisons, *comp* stops comparing and displays the following message:

10 Mismatches - ending compare

If the file sizes are different, *comp* displays the following message:

Files are different sizes, do you wish to continue (Y/N)?

You can either continue the comparison or end it. If you choose to continue, *comp* compares the files until it reaches the end of the shorter file.

After a successful comparison, *comp* displays the following message:

Files compare OK

After the comparison of the two files ends, *comp* proceeds with the next pair of files that match the two pathname options, until no more files can be found that match the first pathname option. Then *comp* displays the following message:

Compare more files (Y/N)?

At this prompt, you can either compare two more files or end the comparison. If you want to compare two more files, press <y> (for yes). Comp prompts you for two new path options.

For all file comparisons, *comp* first ensures that both files include end-of-file (<CTL>-<z>) marks. If they do not, *comp* displays this message:

EOF mark not found

Examples

In the following example, *comp* compares each file with the extension *.asm* in the current directory on drive C with each file of the same name (but with an extension of *.bak*) in the current directory on drive B:

comp c:*.asm b:*.bak

copy

copies and appends files

Syntax

copy [*drive*:][*pathname*] [*drive*:][*pathname*][/**v**][/**a**][/**b**]

(to copy files)

or

copy [*drive*:][*pathname*][/**v**][/**a**][/**b**] [*drive*:][*pathname*]

copy *pathname* + *pathname*[...]*pathnameN*

(to append files)

Description

If you do not specify the second pathname, the copy is created on the default drive and has the same name as the original file (first pathname). If the original file is on the default drive and you do not specify the second pathname, *copy* quits (you are not allowed to copy a file to itself), and MS-DOS displays the following error message:

```
File cannot be copied onto itself
0 File(s) copied
```

If the source and target files are both in the working directory, you may use filenames instead of complete pathnames.

The second *drive*: and *pathname* options may take one of three forms:

- If the second option is a drive name only, MS-DOS copies the original file to the designated drive, keeping the original filename. For example, the following command makes a copy on drive B named *memo.doc*:

  copy memo.doc b:

- If the second option is a filename only, MS-DOS copies the original file to one on the default drive, and renames it with the specified filename. For example, the following command makes a copy of *memo.doc*, names it *letter.doc*, and places it on the default drive:

  copy memo.doc letter.doc

- If the second option includes a drive name, MS-DOS copies the original file to one on the specified drive. For example, the following command makes a copy of *memo.doc* on the default drive, names the copy *letter.doc*, and places the copy on the disk in drive B:

  copy memo.doc b:letter.doc

The */v* switch causes MS-DOS to verify that the sectors written on the target disk are recorded properly. If MS-DOS cannot verify a write, it displays an error message. Although there are rarely recording errors when you run *copy*, the */v* switch lets you verify that critical data has been correctly recorded; it also makes the *copy* command run more slowly because MS-DOS must check each entry recorded on the disk.

The */a* or */b* switch lets you copy either ASCII or binary files, respectively. Each switch applies to the filename preceding it, and to all remaining filenames in the command, until *copy* encounters another */a* or */b* switch.

Examples

When used with a source filename:

/a Causes the file to be treated as an ASCII (text) file. Data in the file is copied up to but not including the first end-of-file mark (in *edlin* this is <CTL>**z**). The remainder of the file is not copied.

/b Causes the entire file to be copied, including any end-of-file marks.

When used with a target filename:

/a Causes an end-of-file character to be added as the last character of the file; for example:

copy memo.doc /a letter.doc

/b Does not add an end-of-file character; for example:

copy billing.asm /b billing2.asm

When you are appending files the default switch is always */a*.

The *copy* command also allows you to append files. To do this you simply list any number of files as options to *copy*, each separated by a plus sign (+), then specify a target file to send the combined files to; for example:

copy intro.rpt + body.rpt + b:sum.rpt report

This command combines files named *intro.rpt*, *body.rpt*, and *sum.rpt* (on drive B), and places them in a file called *report* on the default drive. If you leave out the target file, MS-DOS combines the files into the first specified file.

You can also combine several files into one by using wildcards; for example:

copy *.txt combin.doc

This command takes all files with an extension of *.txt* and combines them into one file named *combin.doc*.

In the following example, each file that matches **.txt* is combined with its corresponding *.ref* file. The result is a file with the same filename but with the extension *.doc*. Thus, *file1.txt* is combined with *file1.ref* to form *file1.doc*, *xyz.txt* with *xyz.ref* to form *xyz.doc*, and so on:

copy *.txt + *.ref *.doc

The following command combines all files matching **.txt* and all files matching **.ref* into one file named *combin.doc*:

copy *.txt + *.ref combin.doc

Do not try to append files if one of the source filenames has the same extension as the target. For example, if the file *all.txt* already exists, the following command is an error:

copy *.txt all.txt

MS-DOS would not detect the error until it tried to append *all.txt*. But at that point, *copy* might have already destroyed the *all.txt* file.

copy compares the filename of the input file with the filename of the target. If they are the same, that one input file is skipped, and MS-DOS prints the error message "Content of destination lost before copy." Further joining proceeds normally. For example, the following command appends all **.txt* files (except *all.txt*) to *all.txt*:

copy all.txt + *.txt

This command will not produce an error message.

If you want to copy files and subdirectories, you should use *xcopy*. Refer to *xcopy* in this appendix for more information on how to do this.

Notes

copy is an internal command.

ctty

changes the tty (device) from which you issue commands

Syntax

ctty *device*

Comments

The *device* parameter specifies the device from which you are giving commands to MS-DOS. The *ctty* command is useful if you want to change the device on which you are working. In this command, the letters *tty* represent the console; that is, your keyboard.

Examples

The following command moves all command I/O (input/output) from the current device (the console) to an AUX port such as another terminal:

```
ctty aux
```

The next command moves I/O back to the console:

```
ctty con
```

Notes

There are many programs that do not use MS-DOS for input, output, or either. These programs send input directly to the hardware on your computer. The *ctty* command has no effect on these programs; it affects only programs that use MS-DOS.

ctty is an internal command.

date

enters or changes the date known to the system

Syntax

date [*mm-dd-yy*]

Description

You can change the date from your terminal or from a batch file. (MS-DOS does not automatically display a prompt for the date if you use an *autoexec.bat* file, so you may want to include a *date* command in that file.) MS-DOS records this date in the directory when you create or change a file.

If your system contains a CMOS clock, that clock date can also be changed using *date*.

Remember to use only numbers when you type the date; allowed numbers are:

mm = 1 – 12
dd = 1 – 31
yy = 80 – 79 or 1980 – 2079

The date, month, and year entries may be separated by hyphens (–) or slashes (/). MS-DOS is programmed to change months and years correctly, whether the month has 31, 30, or 28 days – or 29 days, since MS-DOS handles leap years, too.

You can change the *mm-dd-yy* format in which the date is displayed and entered. The *country* command in the *config.sys* file allows you to change the date format to the European standard *dd-mm-yy*. For more information on the *config.sys* file, see Appendix A, "How to Configure Your System".

Examples

If you simply enter *date*, MS-DOS displays the following message:

```
Current date is weekday mm-dd-yy
Enter new date (mm-dd-yy):_
```

If you do not want to change the date shown, press <Return>. Alternatively you can type a particular date immediately after *date*, as in the following example:

date 3-9-86

In this case the "Enter new date:" prompt does not appear after you press <Return>.

Notes

date is an internal command.

del, erase

deletes files

Syntax

del [*drive*:] *pathname*

or

erase [*drive*:] *pathname*

Comments

You can use the * and ? wildcards to delete more than one file at a time.

If the pathname is *.* the prompt "Are you sure?" appears. If you then type y as a response, all files on the disk are deleted.

Once you have deleted a file on your disk, you cannot recover it.

Examples

The following command deletes a file named *vacation*:

del vacation

If you have two files named *vacation.apr* and *vacation.aug*, you can delete them both with the following command:

del vacation.a*

Notes

del/erase is an internal command.

dir

lists the files in a directory

Syntax

dir [*drive*:][*pathname*][/**p**][/**w**]

or

dir

Comments

If you just enter *dir*, all directory entries on the default drive are listed. If you include a drive name in the command, such as *b:*, all entries in the default directory of the disk in drive B are listed. If you include a filename without an extension (*invoices*, for example) MS-DOS lists all files named *invoices* in the default directory of the disk in the default drive.

When you use *dir* with a filename and drive letter (*b:invoices*, for example) MS-DOS displays all files on the disk in drive B with the filename *invoices*. In all cases, (except when using the */w* switch), MS-DOS lists files with their size in bytes and with the time and date of their last modification.

Note that the following *dir* commands are equivalent, since you can use the wildcards **?** and ***** in the pathname option:

COMMAND	EQUIVALENT
dir	**dir *.***
dir filename	**dir filename.***
dir .ext	**dir *.ext**

The switches are:

/p Selects page mode. It also makes the directory display stop scrolling when the screen has filled. To resume scrolling the display, press any key.

/w Selects wide display and causes MS-DOS to display only filenames and not other file information. The wide display lists up to five files per line.

Note that if the *country* command in the *config.sys* file is set to a country other than U.S., the directory date and time formats may differ. For more information on the *config.sys* file, See Appendix A, "How to Configure Your System."

Example

If your directory contains more files than you can see on the screen at one time, enter the following:

dir /p

This command displays the directory one screenful at a time.

Notes

dir is an internal command.

diskcomp

compares the contents of the disk in the source drive to the disk in the target drive

Syntax

diskcomp [*drive*:] [*drive*:] **[/1] [/8]**

Description

The first *drive*: option specifies the drive that contains the source disk, and the second *drive*: option specifies the drive that contains the target disk.

If you specify only one drive, *diskcomp* uses the default drive as the target drive. If you specify the same drive as the source and target, *diskcomp* does a comparison using one drive, and prompts you to insert the disks as appropriate.

The switches are:

/1 Causes *diskcomp* to compare just the first side of the disk, even if the disks and drives that you are using are double-sided.

/8 Causes *diskcomp* to compare just the first 8 sectors per track, even if the disks contain 9 or 15 sectors per track.

diskcomp performs a track-by-track comparison of the disks. It automatically determines the number of sides and sectors per track based on the format of the source disk. If the target disk is not the same type as the disk in the source drive, *diskcomp* displays the following message:

```
Drive types or diskette types not compatible
```

If all the tracks are the same, *diskcomp* displays the message:

```
Compare OK
```

If the tracks are not the same, *diskcomp* displays a "Compare error" message that includes the track and side number where it found the mismatch.

When *diskcomp* completes the comparison, it prompts you with the following message:

```
Compare another diskette (Y/N)?_
```

If you enter **y**, *diskcomp* prompts you to insert the disks and does the next comparison. If you enter **n**, *diskcomp* ends. If the disk in the default drive does not contain MS-DOS and you end *diskcomp*, you'll receive the following message:

```
Insert disk with COMMAND.COM in drive A
and strike any key when ready
```

diskcomp does not work on network drives, and you cannot use it with assigned, joined, or substituted drives. If you attempt to use *diskcomp* with these types of drives, it displays an error message.

When correctly written programs exit back to MS-DOS, they return an exit code: 0 if no error occurred, or a value greater than zero if there was a problem. This exit code can be tested in batch files, and it allows batch programmers to "branch" to an error-handling routine in the batch file.

The *diskcomp* command returns the following exit codes:

0	Compared OK	The disks compared exactly.
1	Did not compare	The disks were not the same.
2	CONTROL-C error	The user terminated with <CTL>**c**.
3	Hard error	An unrecoverable read or write error occurred – did not compare.
4	Initialization error	There is not enough memory – invalid drives or command line syntax.

You can use the batch processing *if* command to perform error processing based on the *errorlevel* returned by *diskcomp*.

Example

If you want to compare two disks, but your computer has only one floppy disk drive, drive A, you can simply enter the following command:

```
diskcomp a:
```

MS-DOS prompts you to insert both disks, as required.

Notes

When comparing a disk with a backup disk that you made with the *copy* command, you may receive the "Compare error" message, even if the files on the disks are identical. This is because the *copy* command duplicates the information, but doesn't necessarily place it in the same location on the target disk.

diskcomp is an external command.

diskcopy

copies the contents of the disk in the source drive to the disk in the target drive

Syntax

diskcopy [*drive*:] [*drive*:] [**/1**]

Description

The first *drive*: option is the source drive. The second *drive*: option is the target drive.

If you omit both options, MS-DOS performs a single-drive copy operation on the default drive. If you omit just the second option, MS-DOS uses the default drive as the target drive. In either case, though, *diskcopy* destroys the contents of the target disk.

The */1* switch allows you to copy only one side of a disk.

The *diskcopy* command works only with floppy disks. You cannot use *diskcopy* with a hard disk.

---

diskcopy prompts you to insert the source and target disks at appropriate times and waits for you to press any key before continuing.

After copying, *diskcopy* then prompts you with the following message:

```
Copy another diskette (Y/N)?_
```

If you press **y**, MS-DOS prompts you to insert source and target disks, and performs the next copy on the drives that you originally specified.

To end the *diskcopy* process, press **n**.

Because disk space is not allocated sequentially, disks that have had a lot of files created and deleted on them become fragmented. So, the first free sector found by *diskcopy* becomes the next sector allocated, regardless of its location on the disk.

A fragmented disk can delay finding, reading, or writing a file. To prevent further fragmentation, you should use either *copy* or *xcopy* to copy your disk, instead of using *diskcopy*. Because *copy* and *xcopy* copy files sequentially to a disk, the new disk will not be fragmented.

The following command, for example, copies all files from the disk in drive A to the disk in drive B:

xcopy a:*.* b:

diskcopy figures out the number of sides to copy, based on the source drive and disk.

When correctly written programs exit back to MS-DOS, they return an exit code: 0 if no error occurred, or a value greater than zero if there was a problem. This exit code can be tested in batch files, and it allows batch programmers to "branch" to an error-handling routine in the batch file.

0 Copied successfully	
1 Non-fatal read/write error	An unrecoverable but non-fatal read or write error occurred.
2 CONTROL-C error	The user entered <CTL>c to terminate *diskcopy*.
3 Fatal hard error	*diskcopy* was unable to read the source disk or format the target disk.
4 Initialization error	There is not enough memory – invalid drives or command line syntax.

You can use the batch processing *if* command to perform error processing based on the *errorlevel* returned by *diskcopy*.

Example

To copy the disk in drive A to the disk in drive B, enter the following command:

diskcopy a: b:

diskcopy prompts you to insert both disks and press any key to begin copying.

Notes

diskcopy is an external command.

exe2bin

converts exe (executable) files to binary format

Syntax

exe2bin [*drive*:] *pathname* [*drive*:] *pathname*

Description

The *exe2bin* command converts *.exe* (executable) files to binary format. The first *pathname* is the input file; if you do not specify an extension, it defaults to *.exe*. The input file is converted to a *.bin* file format (a memory image of the program) and placed in the output file (the second *pathname*).

If you do not specify a drive name, *exe2bin* uses the drive of the input file. Similarly, if you do not specify an output filename, *exe2bin* uses the input filename. And finally, if you do not specify a filename extension in the output filename, *exe2bin* gives the new file the extension *.bin*.

Some restrictions do apply when you use *exe2bin*: the input file must be in valid *.exe* format produced by the linker; the resident, or actual code and data part of the file must be less than 64 Kbytes; and there must be no STACK segment.

With *exe2bin*, two kinds of conversions are possible, depending on whether the initial CS:IP (Code Segment:Instruction Pointer) is specified in the *.exe* file:

- If the CS:IP is not specified in the *.exe* file, *exe2bin* assumes you want a pure binary conversion. If segment fixups are necessary (that is, if the program contains instructions requiring segment relocation), the command *E* prompts you for the fixup value. This value is the absolute segment at which the program is to be loaded. The resulting program will be usable only when loaded at the absolute memory address specified by your application. The command processor will not be able to load the program.

- If the CS:IP is 0000:100H, *exe2bin* assumes that the file will run as a *.com* file with the location pointer set at 100H by the assembler statement ORG (the first 100H bytes of the file are deleted). No segment fixups are allowed, since *.com* files must be segment relocatable; that is, they must assume the entry conditions explained in the Microsoft Macro Assembler manuals (*User's Guide* and *Reference Manual*). Once the conversion is complete, you may rename the output file with a *.com* extension. The command processor will

then be able to load and execute the program in the same way as the *.com* programs supplied on your MS-DOS disk.

Notes

exe2bin is an external command.

exit

exits the *command.com* program (the command processor) and returns to a previous level, if one exists

Syntax

exit

Description

If you use the MS-DOS *command* program to start a new command processor, you can use *exit* to return to the old command processor. Also, while running an application program, you can exit to the MS-DOS command processor, and then return to your program.

Refer to *command* in this chapter for more information.

Example

If you start a new command processor by entering:

command c:

you can return to the previous command processor by entering

exit

Notes

exit is an internal command.

fastopen

reduce the time needed to open frequently-used files

Syntax

fastopen [*drive*:[=*nnn*][...]]

Description

The *fastopen* command is an MS-DOS kernel extension that remembers the locations of files on fixed disks. Whenever *fastopen* is installed and a file is to be opened, *fastopen* is asked for the location of the file. If the location of the file is known, *fastopen* returns the location. The file can then be opened with greatly reduced disk activity. If *fastopen* is not installed, the normal file opening method is used. Every time a file is opened, *fastopen* records the name and location of the file.

The *fastopen* command only works with local, fixed disks. It can work with up to four fixed disks at a time. The *fastopen* command works with the disks corresponding to the drive specifications included in its command line arguments. The *fastopen* command does not work with remote drives.

By default, the last 10 files opened on each disk are remembered. As few as 10 and as many as 999 files per disk can be remembered by using an optional argument along with a drive specification. As an example, the following line will cause *fastopen* to track 100 files on drive C:.

fastopen c:=100

Approximately 40 bytes of extra memory is needed to record information on each additional file.

fc

Compares two files or two sets of files and displays the difference between them.

Syntax

For ASCII comparisons:

fc [**/a**][**/c**][**/L**][**/Lb n**][**/n**][**/t**][**/w**][**/nnnn**][*drive*:]*pathname* [*drive*:] *pathname*

For binary comparisons:

fc [**/b**][**/nnnn**][*drive*:]*pathname* [*drive*:]*pathname*

Description

The *fc* command matches the first file against the second and reports any differences between them.

The switches that you can use with the *fc* command are described below:

/a Abbreviates the output of an ASCII comparison. Instead of displaying all the lines that are different, *fc* displays only the lines that begin and end each set of differences.

/b Forces a binary comparison of both files. The *fc* command compares the two files byte-by-byte, with no attempt to resynchronize after a mismatch. The mismatches are printed as follows:

xxxxxxxx: yy zz

(where *xxxxxxxx* is the relative address from the beginning of the file of the pair of bytes). Addresses start at 00000000; *yy* and *zz* are the mismatched bytes from the first pathname and second pathname, respectively. The */b* switch is the default when you compare *.exe, .com, .sys, .obj, .lib,* or *.bin* files.

/c Causes the matching process to ignore the case of letters. The *fc* command then considers all letters in the files as uppercase letters.

/L Compares the files in ASCII mode. This switch is the default when you compare files that do not have extensions of *.exe, .com, .sys, .obj, .lib,* or *.bin.*

/Lb n Sets the internal line buffer to *n* lines. The default length of the internal buffer is 100 lines. Files that have more than this number of consecutive differing lines, will abort the comparison.

/n Displays the line numbers on an ASCII comparison.

/t Does not expand tabs to spaces. The default is to treat tabs as spaces to 8-column positions.

/w Causes *fc* to compress white space (tabs and spaces) during the comparison. If a line contains many spaces or tabs in a row, these characters are considered as single white space.

Note that although *fc* compresses white space, it does not ignore it. The two exceptions are beginning and ending white spaces in a line, which are ignored.

/nnnn Specifies the number of lines that must match after *fc* finds a difference between files. If the number of matching lines in the files is less than this number, *fc* displays the matching lines as differences.

The *fc* command reports differences between two files by displaying the first filename, followed by the lines that differ between the files, followed by the first line to match in both files. The *fc* command then displays the name of the second file followed by the lines that are different, followed by the first line that matches.

The default value for the number of lines to match between files is 2. If you want to change this default, specify the number of lines with the */nnnn* switch.

The *fc* command uses a large amount of memory (enough to hold one hundred lines) as buffer storage space to hold the text files. If these files are larger than available memory, *fc* compares what it can load into the buffer space. If it doesn't find a match in those portions of the files in the buffer space, *fc* stops and displays the following message:

resynch failed. Files are too different

For binary files larger than available memory, *fc* compares both files completely, overlaying the portion in memory with the next portion from disk. All differences are output in the same manner as for those files that fit completely in memory.

Examples

Suppose you want to compare two text files called *monthly.rpt* and *sales.rpt*. To make this comparison, you would simply type the following command line:

fc /a monthly.rpt sales.rpt

If you want to check for differences in files that are not ASCII text files, you can use the */b* switch to force a binary comparison of the files. For example, If you have two executable program files called *profits.exe* and *earnings.exe*, and you want to find out whether they are the same, you could type the following:

fc /b profits.exe earnings.exe

The output from this command line might be similar to the following:

```
00000002:     fc b6
00000004:     12 14
0000000e:     56 92
00000012:     e8 5c
00000013:     bb 7c
00000014:     14 0e
00000015:     0a 0d
0000001e:     43 7e
0000001f:     09 0a
00000022:     be e6
        .
        .
        .
000005e0:     00 61
000005e1:     00 73
000005e2:     00 73
000005e3:     00 69
000005e4:     00 67
000005e5:     00 6e
000005e6:     00 6d
000005e7:     00 65
000005e8:     00 6e

fc: earnings .exe longer than profits.exe
```

If the *profits.exe* and *earnings.exe* files were identical, *fc* would display the following message:

fc: no differences encountered

fdisk

configures hard disks for MS-DOS

Syntax

fdisk

Description

The *fdisk* command configures a hard disk for use with MS-DOS. Before you can use your hard disk for the first time, you must use *fdisk* to configure it. (Your dealer may already have done this step.)

The *fdisk* command displays a series of menus to help you partition your hard disk for MS-DOS.

Notes

fdisk is an external command.

find

searches for a specific string of text in a file or files

Syntax

find [/v][/c][/n] "*string*" [[*drive*:][*pathname*]...]

Description

The *find* command is a filter that takes as options a string (a group of characters) and a series of filenames. After searching the files given on the command line, *find* displays any lines it has found that contain the specified string. In your command line, this string must be enclosed in double quotation marks.

If you do not specify a pathname, *find* takes input from its standard input (usually the keyboard) and displays any lines that contain the specified string.

The switches are:

/v Displays all lines not containing the specified string.

/c Prints only the count of lines that contain a match in each of the files.

/n Precedes each line by its relative line number in the file.

Examples

The following command displays all lines from a file named *pencil.ad* that contain the string "Automatic Pencil Sharpener":

find "Automatic Pencil Sharpener" pencil.ad

Notes

You must type double quotation marks around a string that already has double quotation marks.

find is an external command.

format

formats the disk in the specified drive to accept MS-DOS files

Syntax

format *drive*:**[/1][/4][/8][/n:***xx***][/t:***yy***][/v][/s]**

format *drive*:**[/1][/b][/n:***xx***][/t:***yy***]**

Description

The *format* command initializes the directory and the file allocation tables on a disk. You must use this command to format all new disks before MS-DOS can use them.

When using the commands, you must specify the drive that you want to format. The *format* command then uses the drive type to determine the default format for a disk.

When you format a hard disk, *format* prompts you to verify the volume label:

```
Enter current Volume Label for drive x:
```

If your hard disk does not have a volume label, press <Return>. (Note: If your hard disk has never been formatted before, or if it has a bad boot sector, *format* will not prompt you for a volume label.)

If the volume label that you enter does not match the label on the hard disk, *format* displays the following message:

```
Invalid Volume ID Format failure
```

Otherwise, it continues:

```
WARNING, ALL DATA ON NON-REMOVABLE DISK
DRIVE x: WILL BE LOST! Proceed with Format (Y/N)?_
```

If you want to format your hard disk, enter **y**. If not, enter **n**.

The switches are:

/1 Formats a disk for single-sided use, even if the disk or drive is double-sided. If the drive is double-sided and you don't specify this switch, you won't be able to use the formatted disk in a single-sided drive.

/4 Formats a double-sided disk in a high-capacity disk drive. Note, however, that if you are using a single-sided or double-sided drive, you may not be able to reliably read disks formatted with this switch.

/8 Formats a disk for 8 sectors per track. If you do not specify this switch, *format* defaults to either 9 or 15 sectors per track (depending on the type of drive being used). Note that *format* always creates either 9 or 15 sectors per track; when you specify this switch, though, it tells MS-DOS to use only 8 sectors per track.

/b Formats the disk, leaving ample space to copy an operating system, such as MS-DOS 3.3.

/n:xx Specifies the number of sectors per track that *format* uses to format a floppy disk.

/t:yy Specifies the number of tracks that *format* places on a floppy disk.

/v Prompts for a volume label after the disk is formatted. A volume label identifies the disk and can be up to 11 characters in length. An example of a volume label is PROGRAMS.

/s Must be the last switch that you type. This switch copies the operating system files from the disk in the default drive to the newly formatted disk. The files are copied in the following order:

io.sys
msdos.sys
command.com

If the operating system is not on the default drive, *format* prompts you to insert a system disk in the default drive (or in drive A if the default drive is nonremovable).

When formatting is complete, *format* displays a message showing the total disk space, any space marked as defective, the total space used by the operating system (when you use the */s* switch), and the space available for your files.

The following table shows which switches you can use for certain types of disks:

DISK TYPE	VALID SWITCHES
160/180 Kbytes	*/1 /4 /8 /b /n /t /v /s*
320/360 Kbytes	*/1 /4 /8 /b /n /t /v /s*
720 Kbytes	*/n /t /v /s*
1.2 Mbytes	*/n /t /v /s*
hard disk	*/v /s*

You should not format disks on drives used in *assign*, *join*, or *subst*, and you cannot format disks over a network.

The *format* command returns the following exit codes:

0 Successful completion

3 Terminated by user (<CTL>**c**)

4 Fatal error (any error other than 0, 3, or 5)

5 No response to hard disk prompt, "Proceed with format (Y/N)?"

You can check these exit codes by using the *errorlevel* condition with the *if* batch processing command.

Examples

To format a floppy disk in drive A and put the operating system on it, enter the following:

format a: /s

And to format a floppy disk in drive A for use with data, enter the following:

format a: /v

The */v* switch causes *format* to prompt you for a volume label. You should type a label since it will help you identify the data that the disk contains.

Notes

Formatting destroys any previously existing data on a disk and ignores drive assignments created with the *assign* command.

format is an external command.

graftabl

loads graphics character data into a table in memory for use with a color or graphics adapter

Syntax

graftabl [*xxx*]

or

graftabl */status*

Description

The *graftabl* command loads a table of the ASCII characters, 128 through 255, into memory. If you have a color or graphics adapter, this table lets you display foreign language characters when you are in graphics mode.

The *xxx* option is a code page identification number. Valid code pages include the following:

437	United States (default)
850	Multilingual
860	Portuguese
863	French-Canadian
865	Nordic

If you type the *graftabl* command followed by the */status* switch, MS-DOS displays the active character set. After *graftabl* loads the character table, it displays the following message:

GRAPHICS CHARACTERS LOADED

You can load the graphics table only once each time you start MS-DOS. You can put the *graftabl* command in your *autoexec.bat* file. If you try to load the table a second time, *graftabl* displays the following message:

GRAPHICS CHARACTERS ALREADY LOADED

Example

To load the graphics table into memory, enter the following:

graftabl

Notes

This command increases the size of MS-DOS resident in memory.

graftabl is an external command.

For more information about using code pages, see the *chcp* command in this Appendix.

graphics

prints a graphics display screen

Syntax

graphics [*printer*] [**/b**][**/p=port**][**/r**][**/lcd**]

Description

To use *graphics*, you must be using a color or graphics monitor adapter. The printer option may be one of the following:

COLOR1	Prints on an IBM Personal Computer Color Printer with black ribbon.
COLOR4	Prints on an IBM Personal Computer Color Printer with RGB (red, green, blue, and black) ribbon.
COLOR8	Prints on an IBM Personal Computer Color Printer with CMY (cyan, magenta, yellow, and black) ribbon.
COMPACT	Prints on an IBM Personal Computer Compact Printer.
GRAPHICS	Prints on an IBM Personal Graphics Printer, or IBM Proprinter.
THERMAL	Prints on an IBM PC-convertible.

If you do not specify the printer option, *graphics* defaults to the GRAPHICS printer type.

The switches are:

/r Prints black and white (as seen on the monitor) on the printer. The default is to print black as white and white as black.

/b Prints the background in color. This option is valid for COLOR4 and COLOR8 printers.

/lcd Allows *graphics* to print screens from the LCD display on the IBM PC portable.

/p=port
Sets the parallel printer port that *graphics* sends its output to when you press the <Shift> and <PrintScreen> key combination. The port may be set to 1, 2, or 3; the default setting is 1.

To print the screen, press the <Shift> and <PrintScreen> keys at the same time. If the computer is in 320x200 color graphics mode, and if the printer type is COLOR1 or GRAPHICS, *graphics* prints the screen contents with up to four shades of gray. If the computer is in 640x200 color graphics mode, *graphics* prints the screen contents sideways on the paper.

Example

To print a graphics screen on your printer, enter the following command:

graphics

Then, when the screen displays the information you want to print, press the <Shift> and <PrintScreen> keys at the same time.

Notes

This command increases the size of MS-DOS resident in memory.

graphics is an external command.

join

joins a disk drive to a specific path

Syntax

join [*drive*: *drive*:*path*]

join *drive*: **/d**

join

Description

With *join* you don't need to name physical drives with separate drive letters. Instead, you can refer to all the directories on a specific drive with one path. If the path already existed before you gave the *join* command, you cannot use it while the "join" is in effect. Also, you cannot join a drive if it is being used by another process.

If the path does not exist, MS-DOS tries to make a directory with that path. After you give the *join* command, the first drive name becomes invalid, and if you try to use it MS-DOS displays the "Invalid drive" error message.

Examples

You can join a drive only with a root level directory. For example, this command will work:

join d: c:\sales

But the following one will not:

join d: c:\sales\regional

To reverse *join* ("unjoin") use the following format:

join *drive*: **/d**

Here *drive*: represents the source drive, and the */d* switch turns off *join*.

If you just enter *join*, MS-DOS displays the current drives that are joined.

Notes

join is an external command.

keyb

loads a keyboard program

Syntax

keyb [*xx*[,[*yyy*],[[*drive*:][*path*]*filename*]]]

or

keyb

Description

The *xx* option is one of the following two letter country codes:

CODE	KEYBOARD TYPE	COMMAND
us	United States	*keybus (default)*
fr	France	*keybfr*
gr	Germany	*keybgr*
it	Italy	*keybit*
sp	Spain	*keybsp*
uk	United Kingdom	*keybuk*
po	Portugal	*keybpo*
sg	Swiss-German	*keybsg*
sf	Swiss-French	*keybsf*
dk	Denmark	*keybdk*
be	Belgium	*keybbe*
nl	Netherlands	*keybnl*
no	Norway	*keybno*
la	Latin America	*keybla*
sv	Sweden	*keybsv*
su	Finland	*keybsu*

You should load only one keyboard program after starting MS-DOS.

The *yyy* option is the code page which defines the character set.

The *filename* option is the name of the keyboard definition file.

If you type *keyb* without options, MS-DOS displays a message like the following to show the current keyboard code and its related code page, and the current code pages used by your console screen device (CON):

```
Current keyboard code: FR Code page : 437
Current CON code page : 437
```

The *keyb* command allows you to use characters that are not part of the normal (QWERTY) keyboard format. Using the *keyb* command with one of the two letter codes above, you can type commands or text to MS-DOS using either the standard keyboard or a special keyboard.

Note that the characters that appear on your screen when you type on a standard keyboard do not necessarily match the label on the key. You can produce some characters in the non-United States keyboard sets by pressing <CTL> and <ALT> along with a character key. To produce accented (and umlauted) characters, you press dead keys. Dead keys are keys that do not display a character when used alone, but when followed by a letter, display that letter with an accent.

You can switch from the *keyb* program to the default (United States) keyboard format at any time by pressing <CTL><ALT><F1>. You can then return to the memory-resident keyboard program by pressing <CTL><ALT><F2>.

Example

To use a German keyboard, enter the following command:

```
keybgr
```

Notes

keyb is an external command.

You can include the appropriate *keyb* command in your *autoexec.bat* file so that you won't have to type it each time you start MS-DOS.

label

creates, changes, or deletes the volume identification label on a disk

Syntax

label [*drive*:][*label*]

Description

The volume label may be up to 11 characters in length and may include spaces, but not tabs. Aside from tabs, you also should not use the following characters in a volume label:

```
* ? / \ | .
, ; : + = < > [ ]
```

If you do not specify a volume label, *label* prompts you with the following message:

```
Volume in drive X is xxxxxxxxxx
Volume label (11 characters, ENTER for none)? _
```

If the disk does not already have a volume label, *label* prompts you with the message:

```
Volume in drive X has no label
Volume label (11 characters, ENTER for none)? _
```

Type the volume label that you want and press <Return>. Or, you can press <Return> immediately if you want to delete the volume label. The *label* command would then prompt you with the message:

```
Delete current volume label (Y/N)?_
```

If you press **y**, *label* deletes the volume label on the disk. Otherwise, the volume label stays the same.

Example

Suppose you have a disk in drive A that contains sales information for 1986. To label this disk you might enter the following:

```
label a:sales1986
```

Notes

You can use the MS-DOS *dir* command to determine if the disk already has a volume label.

label is an external command.

mkdir, md

makes a new directory

Syntax

mkdir [*drive*:]*path*

Description

With this command you can create a multilevel directory structure. For instance, when you are in your root directory, you can create subdirectories. Remember, though, that when you create directories with *mkdir*, they always appear under your working directory unless you explicitly specify a different path with *mkdir*.

Examples

The following command creates a subdirectory named **\user** in your root directory:

mkdir \user

Now, suppose you want to create a directory named **pete** under the **\user** directory. To do this you could simply enter the following command:

mkdir \user\pete

Notes

mkdir is an internal command.

mode

sets operation modes for devices

Syntax

Parallel printer mode:

mode LPT*n*[:][*chars*][,[*lines*][,**p**]]

Asynchronous communications mode:

mode COM*m*[:]*baud*[,*parity*[,*databits* [,*stopbits*[,**p**]]]]

Redirecting parallel printer output:

mode LPT*n*[:]=**COM***m*[:]

Display modes:

mode *display*

mode [*display*],*shift*[,**t**]

Device code page modes:

mode device codepage prepare=[[*yyy*] [*drive*:] [*path*] *filename*]

mode device codepage select=*yyy*

mode device codepage *refresh*

mode device codepage [*/status*]

Description

The *mode* command prepares MS-DOS for communication with devices such as parallel and serial printers, modems, and console screens. It also prepares parallel printers and console screen devices for code page switching. You can use the *mode* command to redirect output.

For parallel printer modes, you can use PRN and LPT1 interchangeably:

n Specifies the printer number: 1, 2, or 3.

chars Specifies characters per line: 80, or 132.

lines Specifies vertical spacing, lines per inch: 6, or 8.

p Specifies that *mode* tries continuously to send output to the printer if a time-out error occurs.

This option causes part of *mode* to remain resident in memory.

The default settings are LPT1, 80 characters per line, and 6 lines per inch.

You can break out of a time-out loop by pressing <CTL><Break>.

For asynchronous communications modes:

m Specifies the asynchronous communications (COM) port number: 1, 2, 3, or 4.

baud Specifies the transmission rate: 110, 150, 300, 600, 1200, 2400, 4800, 9600, or 19,200.

parity Specifies the parity: N (none), O (odd), or E (even). The default value is E.

databits Specifies the number of databits: 7, or 8. The default is 7.

stopbits Specifies the number of stop bits: 1, or 2. If baud is 110, then the default number of stop bits is 2; otherwise, the default is 1 stop bit.

p Specifies that *mode* is using the COM port for a serial printer and continuously retrying if time-out errors occur.

This option causes part of *mode* to remain resident in memory.

The default settings are COM1, even parity, and 7 databits. If baud is 110, then the default number of stop bits is 2; otherwise, the default is 1 stop bit.

For redirecting parallel printer output (to an asynchronous communications port):

n Specifies the parallel printer port number: 1, 2, or 3.

m Specifies the asynchronous communications port number: 1, or 2.

Redirection causes part of the *mode* program to remain resident in memory.

You must use *mode* to specify the asynchronous communications mode before you can redirect parallel printer output to it.

For setting display modes:

display Specifies one of the following values: 40, 80, BW40, BW80, CO40, CO80, or MONO.

40 indicates 40 characters per line.

80 indicates 80 characters per line.

BW and CO refer to a color graphics monitor adapter with color disabled (BW) or enabled (CO).

MONO specifies a monochrome display adapter with a constant display width of 80 characters per line.

shift Specifies the direction that you want to shift the display: R (right), or L (left).

This option causes part of *mode* to remain resident in memory.

t Specifies a test pattern for aligning the display. If you specify t, *mode* asks if the screen is aligned properly. If you type n, *mode* repeats the shift and asks if the screen is aligned properly. The command ends when you type y.

For device code page modes, you can use the command to set or display code pages for parallel printers on your console screen device. You can use the following options with *mode* to set or display code pages:

device Specifies the device to support code page switching. Valid device names are con, lpt1, lpt2, and lpt3.

yyy Specifies a code page. Valid code pages are 437, 850, 860, 863, and 865.

filename Identifies the name of the code page information (.cpi) file MS-DOS should use to prepare a code page for the device specified.

There are four keywords to use with the mode device codepage command. Each causes the *mode* command to perform a different function:

prepare Tells MS-DOS to prepare code pages for a given device. You must prepare a code page for a device before you can use it with that device.

select Specifies which code page you want to use with a device. You must prepare a code page before you can select it.

refresh If the prepared code pages are lost due to hardware or other errors, this keyword reinstates the prepared code pages.

/status Displays the current code pages prepared and, or selected for a device. Note the following both produce the same results:

mode con codepage

mode con codepage /status

Typing */status* is optional.

Examples

If you want your computer to send its printer output to a serial printer, you need to use *mode* twice. The first *mode* command specifies the asynchronous communications modes, and the second *mode* command redirects the computer's parallel printer output to the asynchronous communications port specified in the first *mode* command.

For example, if your serial printer operates at 4800 baud with even parity, and if it is connected to the COM1 port (the first serial connection on your computer), you would enter the following:

```
mode com1:48,e,,,p
mode lpt1:=com1:
```

If you redirect parallel printer output from LPT1 to COM1, and then decide that you want to print a file using LPT1, you can simply enter the following command:

```
mode lpt1:
```

This disables any redirection of LPT1.

Suppose you want your computer to print on a parallel printer that is connected to your computer's second parallel printer port (LPT2). If you want to print with 80 characters per line and 8 characters per inch, you would enter one of the following commands:

```
mode lpt2:80,8
```

or

```
mode lpt2:,8
```

If you want your computer to keep trying to print a file until your printer is ready to print it, enter this next command:

mode lpt2:80,8,p

To stop retrying to print, you can press <CTL><Break> or enter *mode* without the *p* option.

Notes

If you print files every time you start MS-DOS, you may want to include *mode* commands in your *autoexec.bat* file. See Chapter 4, "Batch Processing," for more information on the *autoexec.bat* file.

mode is an external command.

more

sends output to the console one screen at a time

Syntax

more < *source*

or

source | **more**

Description

The *more* command is a filter that reads from standard input (for instance, a command from your terminal) and displays one screen of information at a time. The *more* command then pauses and displays the "--More--" message at the bottom of your screen. To continue displaying information, press the <Return> key and keep pressing it until you have read all the data.

To hold input information until it is displayed, *more* creates a temporary file on the disk. If the disk is full or write-protected, however, *more* will not work.

Examples

The command *more* is useful for viewing long files. For example, if you have a long file of customers, you could use *more* to view it one screen at a time. Suppose this file is called *clients.new*. To see it you would just enter the following command:

type clients.new | more

You can also redirect input from a file to *more*, for example:

more < clients.new

This command also sends the file *clients.new* to the screen one screenful at a time.

Notes

more is an external command.

nlsfunc

switch to a new *country.sys* file to provide National Language Support

Syntax

nlsfunc [[*drive*:][*path*]*filename*]

Description

The *nlsfunc* command is used to switch to a new *country.sys* file, that contains country specific information different from that defined in the *config.sys* file and loaded at system initialization time.

[**d:**][*path*]*filename*[*.ext*]
specifies the name and location of the file that contains country information. The default value for this parameter is the location and name of the file name defined by the *country* command.

path

sets a command search path

Syntax

path [*drive*:][*path*][;[*drive*:][*path*]...]

or

path ;

Description

The *path* command lets you tell MS-DOS which directories to search for external commands after it searches your working directory. The default value is no path.

For instance, to tell MS-DOS to search the *\bin* directory for external commands, you would simply enter *path* followed by the directory name *\bin*. Then, until you exit MS-DOS or set another path, MS-DOS searches the *\bin* directory for external commands.

You can tell MS-DOS to search more than one path by specifying several paths separated by semicolons. If you use *path* without options, it prints the current path. And if you use the following command, MS-DOS sets the NUL path:

path ;

Example

The following command tells MS-DOS to search three directories to find external commands (the three paths for these directories are *\user\pete*, *b:\user\emily*, and *\bin*):

path \user\pete;b:\user\emily;\bin

MS-DOS searches the paths in the order specified in *path*.

Notes

path is an internal command.

print

prints a file while other MS-DOS commands are being processed

Syntax

print [**/d:***device*][**/b:***size*][**/u:***value*][**/m:***value*][**/s:***timeslice*][**/q:***qsize*] [**/t**][**/c**][**/p**] [*drive***:**][*pathname*]

Description

You can use *print* only if you have an output device, such as a printer or plotter, attached to your computer serial or parallel port.

The following switches are allowed only the first time you run the print command after starting MS-DOS:

/d:device Specifies the *print* device name. If you do not give this switch, *print* prompts for a print device. If you do not specify one, the default device is PRN (the first parallel printer connected to your computer).

Other possible print device names are AUX, LPT1, LPT2, LPT3, COMx, where *x* is a number from 1 to 4 and refers to a serial port. These represent the first, second, and third parallel printers, and the first, second, third and fourth serial printers connected to your computer.

/b:size Sets the size in bytes of the internal buffer. To speed up the *print* command, you increase the value of */b*. The minimum value is 512, the maximum value is 16,386.

/u:value Specifies the number of clock ticks *print* will wait for a printer. If the printer is not available within the time specified, the job will not run. The default for value is 1.

/m:value Specifies the number of clock ticks *print* can take to print a character on the printer. Values range from 1 to 255. The default value is 2.

/s:timeslice The interval of time to be used by the MS-DOS scheduler for the *print* command.

/q:size Specifies the number of files allowed in the print queue, if you want more than 10. The minimum value for the */q* switch is 4, the maximum 32, and the default, 10. To

change this default number of files, you must use the *print* command without any filenames; for example:

print /q:32

/t Deletes all files in the print queue (those files waiting to be printed).

/c Turns on cancel mode and removes the preceding filename and all following filenames from the print queue.

/p Turns on print mode and adds the preceding filename and all following filenames to the print queue.

The *print* command, when used without options, displays the contents of the print queue on your screen without affecting the queue.

Examples

The following command empties the print queue for the device named LPT1:

print /t /d:LPT1

The following command removes the *pencil.tst* file from the default print queue:

print a:pencil.tst /c

The next two commands show how to remove the file *pencil.tst* from the queue and then add the file *pen.tst* to the queue:

print pencil.tst /c
print pen.tst /p

Notes

Each print queue entry may contain a maximum of 64 characters, including the drive name. So if you want to print files in deep subdirectories (many levels down), you may need to change directories.

print is an external command.

prompt

changes the MS-DOS command prompt

Syntax

prompt [[*text*][$*character*]...]

Description

The *prompt* command lets you change the MS-DOS system prompt (for example, A>). If, when using *prompt*, you do not type a new value, the prompt is set to the default value, which includes the default drive designation.

You can use the characters in *prompt* to create special prompts. You must precede each character with a dollar sign ($):

SPECIFY THIS CHARACTER	TO GET THIS PROMPT
$	The $ character
t	The current time
d	The current date
p	The working directory of the default drive
v	The version number
n	The default drive
g	The > character
l	The < character
b	The | character
_	<Return><Linefeed>
e	ASCII code X'1B' (escape)
q	The = character
h	Backspace (to erase a character that has been written to the prompt line)

Examples

The following example sets the drive prompt to *drive:current_directory*:

prompt $p

The following command sets a two-line prompt that displays the following:

Time = (*current time*)
Date = (*current date*)

prompt time = t_date = $d

If your terminal has an ANSI escape sequence driver, you can use escape sequences in your prompts. The following command, for example, sets your prompt in inverse video mode and returns to video mode for other text:

prompt $e[7m$n:$e[m

Notes

prompt is an internal command.

recover

recovers a file or disk containing bad sectors

Syntax

recover [*drive*:]

or

recover [*drive*:][*path*]*filname*

Description

If *chkdsk* shows that a sector on your disk is bad, you can use *recover* to recover the entire disk or just the file containing the bad sector.

This action causes MS-DOS to read the file sector by sector, and to skip the bad sectors. When MS-DOS finds a bad sector, it marks the sector so that it no longer allocates your data to that sector.

Examples

To recover a disk in drive A you would enter the following command:

recover a:

Suppose you have a file named *pencil.ad* that has a few bad sectors. To recover this file you would enter the following command:

recover pencil.ad

Notes

recover is an external command.

ren

renames a file

Syntax

rename [*drive*:][*path*]*filename filename*

or

ren [*drive*:][*path*]*filename filename*

Description

The *ren* command renames all files matching the first filename.

You may use wildcards (* or ?) in either filename option, but if you use them in the second filename, *ren* will not change the positions of the corresponding character.

Examples

The following command changes the extension of all filenames ending in *.txt* to *.doc*:

ren *.txt *.doc

In the next example, *ren* renames a file named *chap10* (on drive B) to *part10*:

ren b:chap10 part10

The newly renamed file *part10* remains on drive B.

Notes

ren/rename is an internal command.

replace

replaces files

Syntax

replace [*drive*:]*pathname* [*drive*:][*pathname*][/**a**][/**p**][/**r**][/**s**][/**w**]

Description

The *replace* command lets you easily update files on your hard disk with new versions of software.

The *replace* command performs two functions:

- By default, it replaces files in the target directory with files in the source directory that have the same name. You may use wildcards in source filenames.
- When you specify the */a* switch, *replace* adds files that exist in the source directory (but not in the target directory) to the target directory.

The switches are:

/a Adds new files to the target directory instead of replacing existing ones. You cannot use this switch with the */s* switch.

/p Prompts you with the following message before it replaces a target file or adds a source file:

Replace filename? (Y/N)_

/r Replaces read-only files as well as unprotected files. If you do not specify this switch, any attempt to replace a read-only file causes an error and stops the replace process.

/s Searches all subdirectories of the target directory while it replaces matching files. This switch is incompatible with the */a* switch. The *replace* command never searches subdirectories in the source path.

/w Waits for you to hit any key before it replaces any files. If you do not specify this switch, *replace* begins replacing or adding files immediately.

As files are replaced or added, *replace* displays the filenames on the screen; then at the conclusion of the *replace* operation, it displays a summary line:

NNN file(s) added/replaced

or:

No files added/replaced

Examples

Suppose your hard disk, drive C, contains several old files named *phones.cli* that contain client names and phone numbers. To update these files and replace them with the latest version of the *phones.cli* file on the disk in drive A, you would enter the following command:

replace a:\phones.cli c:\ /s

This command replaces every file on drive C that is named *phones.cli* with the file *phones.cli* from the root directory on drive A.

Suppose you want to add some new printer device drivers to a directory called *c:\mstools*, which already contains several printer driver files for a word processor. To do this, you would enter the following:

replace a:*.prd c:\mstools /a

This command searches the default directory of drive A for any files that have the extension *.prd* (that don't currently exist in the *\mstools* directory on drive C) and then adds these files to *c:\mstools*.

Upon completion, *replace* returns one of the following exit codes:

0	Command successful
1	Command line error
2	File not found
3	Path not found
5	Access denied
8	Insufficient memory
15	Invalid drive
Other	Standard MS-DOS error

You can test for these codes by using the *errorlevel* condition of the batch processing *if* command.

Notes

You cannot use *replace* to update hidden files or system files.

replace is an external command.

restore

restores files that were backed up using the Microsoft or the IBM backup program

Syntax

restore [*drive*:] [*drive*:][*pathname*][/**s**][/**p**][/**a**:*date*][/**b**:*date*][/**e**:*time*] [/**L**:*time*][/**m**][/**n**]

Description

The *restore* command can restore files from disks of different media types. For example, you can restore files from:

- Hard disk to floppy disk
- Floppy disk to floppy disk
- Floppy disk to hard disk
- Hard disk to hard disk

The first *drive*: option is the drive name for the disk containing the backed up files. The second *drive*: and *pathname* options are the drive and pathname of the files you want to restore.

This *restore* program and the one supplied by IBM are compatible except for switches */a:date*, */b:date*, */e:time*, */L:time*, */m*, and */n*, described below.

The switches are:

/s	Restores subdirectories also.
/p	Prompts for permission to restore any hidden or read-only files that match the file specification.
/a:date	Restores only those files that were last modified on or after the given date.
/b:date	Restores only those files that were last modified on or before the given date.
/e:time	Restores only those files that were last modified at or earlier than the given time.

/L:time Restores only those files that were last modified at or later than the given time.

/m Restores only those files that have been modified since the last backup.

/n Restores only those files that no longer exist on the target disk.

Exit codes

0 Normal completion

1 No files were found to restore

3 Terminated by user

4 Terminated due to error

Refer to *backup* in this appendix for more information.

Example

To restore the file *invest.mnt* from the backup disk in drive A to the *\irsharpe* directory on drive C, enter the following:

restore a: c:\irsharpe\invest.mnt

Press <Return> to let MS-DOS know that the backup disk is in drive A. Then, once MS-DOS has restored the file, use the *dir* or *type* command to make sure that the file was restored properly.

Notes

restore is an external command.

rmdir

removes a directory from a multilevel directory structure

Syntax

rmdir [*drive*:]*path*

or

rd [*drive*:]*path*

Description

The *rmdir* command removes a directory that is empty except for the "." and ".." shorthand symbols. Before you can remove a directory entirely, you must delete its files and subdirectories.

Example:

Suppose you want to remove a directory named *\user\pete*. You first issue a *dir* command for the *\user\pete* path to ensure that the directory is empty; then you enter the following command:

```
rmdir \user\pete
```

Notes

rmdir/rd is an internal command.

select

installs MS-DOS on a new floppy disk with desired country-specific information and keyboard layout

Syntax

select [[*drive*:] [*drive*:][*path*]] [*yyy*][*xx*]

Description

The *select* command allows you to install MS-DOS on a new disk with country-specific information (such as date and time formats, and collating sequence) for a selected country.

The *select* command uses *format* to prepare the target disk, then uses *xcopy* to copy the contents of the source disk to the target disk, then creates the *config.sys* and *autoexec.bat* files.

The source drive can be either drive A or B. The default source drive is drive A. The default target drive is drive B.

If you choose a hard disk as the target, MS-DOS will prompt you to type the correct internal label for that disk. If you type the wrong label, *select* ends. When you have typed the correct label, *select* displays a warning like the following:

```
WARNING, ALL DATA ON NON-REMOVABLE DISK
DRIVE D: WILL BE LOST!
Proceed with Format (Y/N)?
```

If you press <n> (for no), *select* ends. If you press <y> (for yes), the target disk is formatted.

You can use the following options with the MS-DOS *select* command:

yyy Specifies the country code. MS-DOS gathers country-specific information such as time and date formats from the *country.sys* file for the country code specified.

xx Specifies the keyboard code for the keyboard layout used. For a list of valid keyboard codes, see the *keyb* command.

Examples

Suppose you want to create a new MS-DOS disk that included the country-specific information and keyboard layout for Germany. With your source disk in drive B and your target disk in drive A, you would type the following:

select b: a: 049 gr

MS-DOS displays this message:

```
SELECT is used to install DOS the first
time. SELECT erases everything on the
specified target and then installs DOS.
Do you want to continue (Y/N)?
```

If the disk in drive A contains any data files, they will be erased, unless you press <n> (for no). If the disk is blank, or reusable, press <y> (for yes) and press the <return> key.

MS-DOS will prompt you to insert a new disk in drive A. After the disk is formatted, MS-DOS copies files from the source disk B to the target disk A.

set

sets one string of characters in the environment equal to another string for later use in programs

Syntax

set [*string*=[*string*]]

Description

You should use *set* only if you want to set values for programs you have written.

When MS-DOS sees a *set* command, it inserts the given string and its equivalent into a part of memory reserved for the environment. If the string already exists in the environment, it is replaced with the new setting.

If you specify just the first string, *set* removes any previous setting of that string from the environment. If you use *set* without options, MS-DOS displays the current environment settings.

When batch processing, you can also use *set* to define your replaceable parameters by name instead of by number. For example, if your batch file contains the statement "type %file%", you could use *set* to set the name that MS-DOS will use for that variable. In the following command, for example, *set* replaces the *%file%* parameter with the filename **taxes.86**:

set file=taxes.86

To change the replaceable parameter names, you don't need to edit each batch file. Note also that when you use text (instead of a number) as a replaceable parameter, the name must be ended by a percent sign.

The *set* command is especially useful in the **autoexec.bat** file, because it lets you automatically set strings or parameters when you start MS-DOS. See Chapter 4, "Batch Processing," for more information about the **autoexec.bat** file.

Example

The following command sets the string "include" to **c:\inc** until you change it with another *set* command:

set include=c:\inc

If you just enter *set*, MS-DOS displays the current environment settings.

Notes

set is an internal command.

share

installs file sharing and locking

Syntax

share [**/f:***space*][**/L:***locks*]

Description

You can see *share* only when networking is active. If you want to install shared files, you can include *share* in your **autoexec.bat** file. To learn more about shared files, see the *Microsoft Networks Manager's Guide*.

MS-DOS has a storage area that it uses to record file sharing information; to allocate file space (in bytes) for this area, you use the */f:space* switch.

Each open file requires enough space for the length of the full filename plus 11 bytes, since the average pathname is around 20 bytes in length. The default value for the */f* switch is 2048.

The */L:locks* switch allocates the number of locks you want to allow. The default value for the */L* switch is 20.

Once you have used *share* in an MS-DOS session, all read and write requests are checked by MS-DOS.

Example

The following example loads file sharing and uses the default values for the */f* and */L* switches:

share

Notes

share is an external command.

sort

sorts data

Syntax

sort [*drive*:][*pathname*][/**r**][/+**n**]

Description

The *sort* command lets you alphabetize a file according to the character in a certain column. You specify the file by the *drive:* and *pathname* options.

The switches are:

/r Reverses the sort, that is, sorts from Z to A.

/+n Sorts the file according to the character in column *n*, where *n* is some number. If you do not specify this switch, the *sort* command sorts the file according to the character in the first column.

Examples

The following command reads the file **expenses.txt**, sorts it in reverse order, and writes the output to a file named **budget.txt**:

sort /r expenses.txt budget.txt

The following command pipes the output of *dir* to *sort*. The *sort* command filter sorts the directory listing starting with column 14 (the column in the directory listing that contains the file size) and sends the output to the screen. The result is a directory, sorted by file size:

dir | sort /+14

The following command does the same thing as the previous one, except that *more* gives you a chance to read the sorted directory one screenful at a time:

dir | sort /+14 | more

Notes

sort does not distinguish between uppercase and lowercase letters.

sort is an external command.

subst

substitutes a path with a drive letter

Syntax

subst [*drive: drive:path*]

subst*drive*: **/d**

Description

The *subst* command lets you associate a path with a drive letter. This drive letter then represents a virtual drive because you can use the drive letter in commands as if it represented an actual physical drive.

When MS-DOS finds a command that uses a virtual drive, it replaces the drive letter with the path and treats that new drive letter as though it belonged to a physical drive.

If you enter *subst* by itself, MS-DOS displays the names of the virtual drives in effect.

To delete a virtual drive you use the */d* switch.

Example

The following command creates a virtual drive, Z, for the pathname **b:\user\betty\forms**:

subst z: b:\user\betty\forms

Note that this example assumes that you have included the line **lastdrive=z** in your *config.sys* file.

Now, instead of typing the full pathname, you can get to this directory by simply typing the name of the virtual drive:

z:

Notes

subst is an external command.

sys

transfers the MS-DOS system files from the disk in the default drive to the disk in the specified drive

Syntax

sys *drive*:

Description

Usually, you use *sys* to update your system or place it on a formatted disk that contains no files. You must type a drive letter with this command.

If the system files **io.sys** and **msdos.sys** are already on the target disk, they must take up the same amount of space on the disk as the new system will need. This means that you cannot transfer system files from an MS-DOS 2.0 disk to an MS-DOS 1.1 disk; instead, before *sys* will work, you must reformat the MS-DOS 1.1 disk with the MS-DOS 2.0 *format* command.

The target disk must be completely blank or must already contain the system files **io.sys** and **msdos.sys**.

The transferred files are copied in the following order:

IO.SYS
MSDOS.SYS

io.sys and **msdos.sys** are both hidden files that do not appear when you enter *dir*.

The *sys* command does not transfer the **command.com** file (the command processor). To transfer **command.com** to the target disk, you must use *copy*.

Notes

sys is an external command.

time

displays and sets the time

Syntax

time [*hours:minutes*]

Description

Time is entered in a 24-hour clock format. If you just enter *time*, the following message is displayed:

```
Current time is hh:mm:ss.cc
Enter new time:_
```

If your system has a CMOS clock you can change it using *time*.

If you don't want to change the time shown, you simply press <Return>. If you want to change the time after you have started MS-DOS (for example, to 8:20 a.m.), enter *time* followed by 8:20 in response to the MS-DOS prompt. Note that letters are not allowed; instead, you must type the time using numbers only.

The options are:

hours = 0-24
minutes = 0-59

Separate the *hour* and *minute* entries by a colon. You do not have to type the *ss* (seconds) or *cc* (hundredths of a second).

Notes

If you do not type a valid time, MS-DOS displays the following message and then waits for you to type a valid time:

```
Invalid time
Enter new time:_
```

As with *date*, you can change *time* format by changing the *country* command in the *config.sys* file.

time is an internal command.

tree

displays the path (and, optionally, lists the contents) of each directory and subdirectory on the given drive

Syntax

tree [*drive:*] [**/f**]

Description

The *tree* command lists the full path of each directory and subdirectory on the specified drive.

The *drive:* option specifies the drive that you want to use. If you do not specify this option, *tree* uses the default drive.

The */f* switch displays the names of the files in each directory.

Example

If you want to print a list of the directory and filenames on the disk in drive B, you can use the following command:

```
tree b: /f > prn
```

Notes

tree is an external command.

type

displays the contents of a text file

Syntax

type [*drive:*]*filename*

Description

To view a text file without modifying it, you can use *type*. (Use *dir* to find the name of a file, and *edlin* to change the contents of a file.)

When you use *type* to display a file that contains tabs, all the tabs are expanded to 8 spaces wide. Also, if you try to display a binary file, you may see strange characters on the screen, including bells, formfeeds, and escape sequences.

Example

If you want to display the contents of a file called **holiday.mar**, you would enter the following command:

type holiday.mar

Notes

type is an internal command.

ver

prints the MS-DOS version number

Syntax

ver

Description

If you want to know what version of MS-DOS you are using, you simply enter *ver*. The version number will then be displayed on your screen.

Example

When you enter *ver*, the following message is displayed:

MS-DOS Version 3.20

Notes

ver is an internal command.

verify

turns the verify switch on or off when writing to a disk

Syntax

verify [on] [off]

Description

You can use *verify* to verify that your files are written correctly to the disk (no bad sectors, for example). MS-DOS performs a *verify* each time you write data to a disk. You will receive an error message only if MS-DOS is unable to successfully write your data to a disk.

Examples

If you want to know the current setting of *verify*, use *verify* without an option:

verify

verify on remains in effect until a program changes it (by a *Set Verify* system call), or until you enter the following:

verify off

This command has the same purpose as the */v* switch in the *copy* command.

Notes

verify is an internal command.

vol

displays the disk volume label or volume ID, if it exists

Syntax

vol [*drive:*]

Description

The *vol* command displays the volume label of the disk in the specified drive. If you do not type a drive letter, MS-DOS displays the volume label of the disk in the default drive.

Example

If you want to see the volume label for a disk in drive A, you could enter the following:

```
vol a:
```

If the volume label is "DOS 3-2", MS-DOS responds by displaying the message:

```
Volume in drive A is DOS 3-2
```

Notes

vol is an internal command.

xcopy

copies files and directories, including lower level directories, if they exist

Syntax

xcopy [*drive*:][*path*][*filename*] [*drive*:][*path*][*filename*]
[/a] [/d:*date*] [/e] [/m] [/p] [/s] [/v] [/w]

Description

The first *drive:*, *path*, and *filename* parameters specify the source file or directory that you want to copy. The second *drive:*, *path*, and *filename* parameters specify the target. You must include at least one of the source parameters. If you omit the target parameters, *xcopy* assumes you want to copy the files to the default directory.

If you do not specify the *path* option, *xcopy* uses the default directory with the default filename, *.*.

The switches are:

/a Copies source files that have their archive bit set. The switch does not modify the archive bit of the source file. See *attrib* for information on how to set the archive attribute.

/d Copies source files that you modified on or after the date specified by *date*. Note that the date format may vary depending on the country code that you are using. See *date* for more information.

/e Copies any subdirectories, even if they are empty. You must use this switch with the */s* switch.

/m Similar to the */a* switch in that it copies archived files only; however, it turns off the archive bit in the source file. See *attrib* for information on how to set the archive attribute.

/p Prompts you with "(Y/N?)" to let you confirm whether you want to create each target file.

/s Copies directories and lower level subdirectories, unless they are empty. If you omit this switch, *xcopy* works within a single directory.

/v Causes *xcopy* to verify each file as it is written to the target to make sure that the target files are identical to the source files.

/w Causes *xcopy* to wait before it starts copying files. *xcopy* displays the following message:

Press any key when ready to start copying files

You must press a key to continue, or press <CTL>**c** to abort the *xcopy* command.

Exit Codes

When correctly written programs exit back to MS-DOS, they return an exit code: 0 if no error occurred, or a value greater than zero if there was a problem. This exit code, which you can test in batch files, lets you "branch" to an error-handling routine in the batch file.

If *xcopy* encounters an error, it returns one of the following exit codes:

0 Copy without error

1 No files found to copy

2 <CTL>**c** entered by user to terminate *xcopy*

4 Initialization error. There is not enough memory – invalid drive or command line syntax, file not found, or path not found.

5 Int 24 error occurred. The user aborted from INT24 error reading or writing disk.

You can test for these codes by using the *errorlevel* condition of the batch processing *if* command.

Examples

Because *diskcopy* copies disks track by track, it requires your source and target disks to have the same format. If you have a disk that contains files in subdirectories and you want to copy it to a target disk that has a different format, you must use *xcopy*. For example, the following command copies all the files and subdirectories (including any empty subdirectories) on the disk in drive A to the disk in drive B:

xcopy a: b: /s /e

The *xcopy* command may prompt you to specify whether the target is a file or a directory. If you don't want to receive this prompt, enter the following command:

copy /b xcopy.exe mcopy.exe

This command creates a new command called *mcopy.exe*. Now you can use *mcopy* the same way you use *xcopy*, except that *mcopy* automatically determines whether the target is a file or a directory.

mcopy uses the following rules for copying files:

- If the source is a directory, the target is a directory.
- If the source includes multiple files, the target is a directory.
- If you append a backslash (\) to the end of the target name, the target is a directory. For example, the following command creates the directory **a:\workers**, if it doesn't already exist, and copies the file **payroll** to it:

  xcopy payroll a:\workers

Notes

xcopy is an external command.

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

ODT-DOS Advanced Topics

ODT-DOS is based on technology developed for Merge 386 by Locus Computing Corporation.

4-13-90-1.0.0C
Processed: Fri May 25 14:50:21 PDT 1990

Preface

About This Book

This book explains the advanced features of ODT-DOS.

Who Should Use This Book

This book is for ODT-DOS users and administrators who want to control the operation of DOS in ways not described in *Using ODT-DOS* and *Administering ODT-DOS*. It covers a wide variety of topics of interest to users who want to tailor their own DOS environments and applications to meet their personal requirements. System administrators can use many of the same techniques to change the default behavior of system resources for all users.

This book assumes you are familiar with *Using ODT-DOS* and *Administering ODT-DOS*. Familiarity with the use and administration of your computer hardware, UNIX® operating system, and DOS commands and applications is also useful for many of the procedures described in this book.

Organization Of This Book

This book has seven chapters. Many of the subjects covered in this book do not require that you understand topics covered earlier in the book. Therefore you don't need to read the chapters in order starting with Chapter 1. If you plan to use the DOS options described in Chapter 7, however, you should first read Chapter 6, Tailoring the Operation of DOS. To find topics of interest, consult the table of contents, the index, or the following outline:

Chapter 1. Using the DOS Environment expands upon the description of the DOS environment in *Using ODT-DOS*, telling you more about standard DOS commands, virtual and real disk drives, and AUTOEXEC.BAT and CONFIG.SYS files. It also describes the merge command, which lets you optimize your DOS environment for use with specific applications.

Chapter 2. Using DOS from the UNIX Shell provides information about using DOS drives, the UNIX search path, and UNIX shell "metacharacters" when you run DOS from the UNIX shell. In addition, it tells you more about using standard DOS commands from the UNIX shell, how the UNIX system interprets DOS command names, and how to

make DOS commands and applications executable from the UNIX shell.

Chapter 3. Advanced Printing describes several printing techniques that are useful under various circumstances.

Chapter 4. Running UNIX Programs from DOS covers many features of the Merge on utilities not described in *Using ODT-DOS* or *Administering ODT-DOS*.

Chapter 5. Other Advanced Topics treats several diverse topics such as expanded memory (EMS), environment variables, and alternative shells.

Chapter 6. Tailoring the Operation of DOS tells you about the commands and files you need to understand in order to tailor the operation of DOS programs and the DOS environment.

Chapter 7. DOS Options is a reference chapter that thoroughly describes each of the DOS options you can use to customize the operation of DOS.

Typographic Conventions

This book uses several typographic conventions to help you recognize what you should type and what ODT-DOS displays. These conventions are:

- References to standard UNIX commands and all commands supplied with Merge are in lowercase bold (for example, **ls -l** or **dos2unix**). References to DOS commands are in uppercase (for example, DIR).

- References to files in the integrated DOS/UNIX file system (which can be used with either DOS or UNIX commands and applications) appear in lowercase bold when they are referred to in a UNIX context and uppercase when they are referred to in a DOS context (for example, **ls /usr/joe/memos/july** or DIR \USR\JOE\MEMOS\JULY).

- Examples showing exactly what you type use the lowercase Courier font for both UNIX and DOS commands:

```
udir /usr/bob
```

- Information printed on your screen is shown either enclosed within a drawing of a CRT screen or, for shorter examples, in Courier bold type. For example:

```
Insert new diskette for drive B:
and strike any key when ready.
```

- Italics indicate generic information for which you should substitute actual values for your system. For example, the following means you should supply the names of the source and target files to be used by the **unix2dos** command:

  unix2dos *source_file target_file*

- Prompts are shown (in bold) in all examples so you know which operating system (UNIX or DOS) the example applies to. When you use the DOS operating system, your prompt is a letter and an angle bracket (such as **C>**). The UNIX prompt is **$**. Examples intended for the system administrator, who should be logged in as **root** or superuser, are shown with a UNIX **#** prompt.

- Examples do not explicitly show carriage returns. It is assumed that you press ENTER at the end of each line.

- References to keys with names such as CTRL, ALT, DEL, BREAK, SHIFT, PRT SC, and SYS REQ appear in uppercase. When you see instructions to press one of these keys, press the indicated key. When two or more of these keys are named with hyphens separating them, it means you should press the keys in the order listed and hold them. For example:

  CTRL-ALT-DEL

  means press and hold the CTRL key, then, while still holding CTRL, press the ALT key, then, while still holding CTRL and ALT, press the DEL key.

- References to standard alphabetic keys on your keyboard are shown in uppercase. Unless explicitly instructed otherwise, do not press the SHIFT key. For example:

  CTRL-D

  means press and hold the CTRL key and then, while holding CTRL, press the D key.

ODT-DOS Books

Other books that describe ODT-DOS operation and administration include:

- *Using ODT-DOS* in the *Open Desktop™ User's Guide.*
- *Administering ODT-DOS* in the *Open Desktop Administrator's Guide.*

Release Notes

Be sure to read the *Open Desktop Release Notes* for up-to-date information on supported hardware and software as well as information on product changes since this book was printed.

Contents

Chapter 1

Using The DOS Environment

This chapter tells you how to take advantage of several advanced ODT-DOS features when you use the DOS environment. It also describes DOS commands that operate differently in the ODT-DOS environment than they do on a conventional DOS computer. This chapter is organized as follows:

- **Exceptional DOS Commands** lists DOS commands that are unavailable in the ODT-DOS environment or operate differently than they do on a stand-alone DOS computer.
- **Using DOS Drives** tells you how to use several specialized ODT-DOS disk drives.
- **Running Copy-Protected DOS Applications** contains hints for running these applications in the ODT-DOS environment.
- **Using the merge Command** describes ways to configure your DOS environment using the **merge** command.
- **Using AUTOEXEC.BAT and CONFIG.SYS Files** tells you about personal and systemwide AUTOEXEC.BAT and CONFIG.SYS files.

Exceptional DOS Commands

Nearly all standard DOS commands operate in the ODT-DOS environment just as they do on a conventional stand-alone DOS computer. The following DOS commands, however, are either not useable in the ODT-DOS environment or operate differently than they do on a stand-alone DOS computer:

- You cannot use the DOS FDISK command. Instead of running FDISK under ODT-DOS, use equivalent UNIX utilities or shut down the UNIX system, boot standard DOS, and use FDISK under standard DOS.
- You can't use SHIP or any other DOS command for parking the fixed disk head on the ODT-DOS system.

- You can't use CHKDSK, FORMAT, or SYS on the shared DOS/UNIX file system. You *can* use these commands on a real DOS file system, such as the diskette drive, a physical DOS partition, virtual floppies, and virtual DOS partitions.

- You *can* use the DOS TIME and DATE commands to display or change the time and date that apply to the DOS environment, but when you leave the DOS environment, time and date are determined by the UNIX clock. When you reenter DOS, the DOS clock is always initially synchronized with the UNIX clock.

If you issue a DOS command that doesn't work in the ODT-DOS environment, DOS displays an error message but doesn't harm your computer in any way or destroy any data.

Using DOS Drives

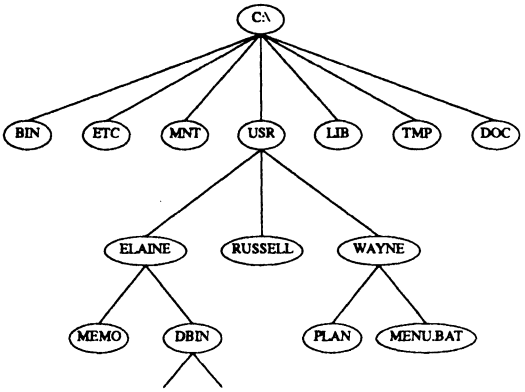
For most purposes, DOS drive C: is the most convenient drive to use when you install and run DOS commands and applications. However, ODT-DOS also has three additional drives, not found on standard DOS computers, that make installing and executing DOS applications more convenient. Drives D: and J: are specialized virtual drives that you can use like DOS drives. Drive E: is a real DOS drive that allows you to use a special section of the fixed disk that is reserved for DOS files.

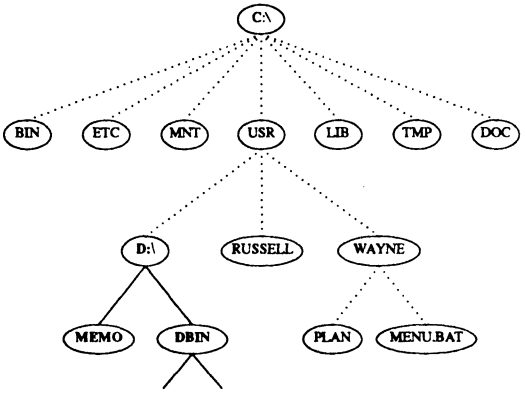
The following sections describe these drives.

Drive D:

Drive D: closely simulates the fixed disk on a stand-alone, single-user personal computer. Your own files and directories on the ODT-DOS fixed disk are accessible on drive D: just as they are on drive C:. On drive D:, however, your home directory is *root*. That is, if you are logged in as the user ELAINE, the directory D:\ contains the same files as C:\USR\ELAINE. Because your home directory is the root of the file system on drive D:, you cannot access any files outside your home directory on drive D:.

The following illustrations show a simplified view of drive C: on a typical ODT-DOS computer and the user Elaine's view of her drive D: on the same computer.





On drive D:, Elaine can access any of her own files (like MEMO) or subdirectories (like DBIN). She cannot, however, access other files and directories on the fixed disk (connected with dotted lines in the illustration) from drive D:. To access these remote files and directories, she must use drive C:.

Drive D: is useful for installing and running some DOS applications that modify or create files in the root directory. When you install such applications on drive D:, they modify or create files in your home directory rather than altering the systemwide root directory (C:\). See *Administering ODT-DOS* for further information on installing DOS applications.

Drive D: is unique for each user.

Drive E:

Drive E: gives you access to the *physical DOS partition*, a special section of the fixed disk that is reserved exclusively for DOS work. Drive E: is usable only if the system administrator has created and formatted the DOS partition. UNIX files cannot be created on drive E: like they can on drives C:, D:, and J:, and UNIX does not have direct access to DOS files created on drive E:. Although drive E: does not share the same files as drives C:, D:, and J:, you use it like a standard DOS disk drive.

Drive E: is used mainly for certain copy-protected DOS applications that cannot be installed on drive C: or drive D:. Once the applications are installed, drive E: is used like any other DOS drive. You run programs from drive E: by making drive E: your current drive or specifying the drive when you invoke the program:

```
C> e:
E> 123
```

Although drive E: is most useful for specific copy-protected applications, you can use it with any DOS commands or applications that you choose.

Drive E: contains no ODT-DOS files when you first install ODT-DOS, but as you use your system, you can add DOS programs, files, and directories to drive E:.

Drive E: is the same for all users. By default, drive E: is a public resource. DOS files and directories created on drive E: are not owned by specific users or protected by UNIX file protection mechanisms. This means that all users can create files on drive E:, and all users have the power to remove or change any file.

Write access to drive E: is available on a first-come, first-served basis. As long as nobody is writing a file on drive E:, all ODT-DOS users can read any file on drive E:. When someone writes to a file on drive E:, no other users can *read or write* to drive E: until the DOS process that is writing on drive E: exits.

For further information on the physical DOS partition, see *Administering ODT-DOS*.

Drive J:

Drive J: works exactly like drive C:, but you can have different current directories on the two drives. When you enter the DOS environment, the current working directory on drive J: is \USR\LDBIN. You can make drive J: your working drive and change directories on drive J: at any time without affecting your current working directory on other DOS drives.

For example, if you want to alternate easily between the directories \USR\DAVE\MEMOS and \USR\WAYNE\MEMOS, define the current directories of drives C: and J: to be these two directories:

```
C> cd c:\usr\dave\memos
C> cd j:\usr\wayne\memos
```

Then you can alternate between the two directories simply by typing c: or j:.

You can also use drive J: to simplify the use of some older DOS applications that require both the application and data or text files to be in the current directory. When you use these applications, make your current directory on one drive (C: or J:) the directory that contains the application, and make your current directory on the other drive (J: or C:) the directory that contains the data or text file.

Virtual DOS Floppies And Virtual DOS Partitions

In addition to drives C:, D:, E:, and J:, your ODT-DOS system may have one or more virtual floppy drives or virtual DOS partitions. Virtual floppies and virtual partitions are files within the shared DOS/UNIX file system that are formatted as DOS volumes. They are not useful in the UNIX environment, but you can use them with DOS as you would use actual physical DOS diskette drives or partitions. *Administering ODT-DOS* describes how you create virtual floppies and partitions.

By default, virtual floppies and partitions are not automatically accessible when you enter the DOS environment. You must use the **dos +a** option to attach any virtual floppies or partitions you want to use during a particular DOS session. The syntax for attaching virtual floppies and partitions is:

+a*drive_letter*:=*pathname*

drive_letter is a or b if you are attaching a virtual floppy. Use any letter between e and z if you are attaching a virtual partition. *pathname* is the full UNIX pathname of the file you want to use as a virtual floppy or partition. For example, to attach the file **/usr/greg/vflop** as drive B:, use the command:

```
$ dos +ab:=/usr/greg/vflop
```

To attach the same file as drive A: and also attach the virtual partition **/usr/greg/vpart** as drive F:, use the command:

```
$ dos +aa:=/usr/greg/vflop +af:=/usr/greg/vpart
```

Once you attach a virtual floppy or partition like this, you can make the attached virtual drive your current working drive and access files just as you would when using a physical DOS diskette or disk drive.

You can also run a specific application and attach a virtual drive with a single command. For example:

```
$ dos +af:=/usr/greg/vpart colorama
```

Note the following points concerning the choice of *drive_letter*:

- Drives C:, D:, E: and J: have specific functions in the ODT-DOS environment, so you should normally avoid using these drive letters to attach virtual partitions.
- The default ODT-DOS LASTDRIVE is n. If you use a drive letter higher than n, you need to redefine LASTDRIVE in your CONFIG.SYS file.

Virtual floppies and partitions have the same access restrictions as the physical DOS partition (drive E:). Multiple DOS processes can read the same virtual floppy or partition at the same time, but when a process writes to the virtual device, no other process can read or write to the device until the writing process exits.

You can use the **dosopt** command as described in Chapter 6 to configure DOS applications or the DOS environment to attach specific virtual floppies or partitions automatically. See the description of the ±a option in Chapter 7 for further information on attaching virtual floppies and partitions.

Reassigning ODT-DOS Drives

Unless you intend to change standard ODT-DOS functionality, do not use the DOS ASSIGN, JOIN, or SUBST commands to redefine drives C:, D:, E: or J: so they refer to other drives or directories. You can, however, use these commands to make other DOS drives refer to the standard ODT-DOS drives without affecting ODT-DOS functionality. For example, the following command defines DOS drive M: so that it refers to the subdirectory REPORTS\MONTHLY within your home directory (D:\):

```
C> subst m: d:\reports\monthly
```

Running Copy-Protected DOS Applications

No special procedures are required to run copy-protected DOS applications on the ODT-DOS system. Simply follow the application manufacturers' instructions for using them. For example, if your application is installed on drive C: but requires a key disk to be in the diskette drive when you run it, use the key disk as you would on a conventional DOS computer.

Some copy-protected applications must be installed on an actual DOS file system. You can install these applications either on the physical DOS partition (drive E:), if your ODT-DOS computer has one, or on a virtual DOS partition. To run an application that is installed on a DOS partition, follow the procedures described earlier in this chapter.

For further information on installing copy-protected DOS applications, refer to *Administering ODT-DOS*.

Using The merge Command

The **merge** command allows you to tune the operation of the DOS process for certain applications. You use the **merge** command to set the following characteristics of the DOS environment from within the environment:

- You can set the clock interrupt rate that ODT-DOS uses.
- You can keep programs that poll the keyboard from being put to sleep.
- You can configure the DOS environment so DOS interprets virtual drives or DOS file handles as being either local or remote.

The syntax of the **merge** command is:

merge set fastclk on|off
merge set pollsleep on|off
merge set drive local|remote
merge set handle local|remote
merge display [*option*]
merge return *option*

These forms of the **merge** command are described below.

fastclk Option

The **merge fastclk** option determines whether ODT-DOS uses the default clock interrupt rate or a lower interrupt rate. The syntax for this option is:

merge set fastclk on|off

The ODT-DOS default is **fastclk on.**

By default, ODT-DOS uses a clock interrupt rate of approximately 18 per second for DOS, which is the same as a conventional DOS computer. You can improve the performance of most DOS commands and applications by using the **fastclk off** option to reduce the interrupt rate to one per second. To reduce the interrupt rate for your current DOS session, type:

```
C> merge set fastclk off
```

Some DOS applications sensitive to the clock interrupt rate do not run correctly at the lower rate. If you run one of these applications with **fastclk** set to **off**, you may notice symptoms such as:

- The application runs much more slowly than expected.
- The application hangs.

If you experience one of these problems and it is caused by the interrupt rate, you can correct it by using the **merge** command to set **fastclk** back to **on.**

To set **fastclk on**, type:

```
C> merge set fastclk on
```

pollsleep Option

The **merge pollsleep** option determines whether programs that poll the keyboard are put to sleep when they are idle. The syntax for this option is:

merge set pollsleep on|off

The ODT-DOS default is **pollsleep on.**

DOS applications that poll the keyboard can consume system resources even when they are idle by entering a polling loop. Applications that enter a polling loop include (but are not limited to) Lotus 1-2-3, dBASE III, and WordPerfect.

If not compensated for, these applications reduce system performance because when they poll the keyboard, less CPU time is available to other concurrently running processes. If you run more than one such application at once, overall system response can degrade to an unacceptable level. By default, ODT-DOS corrects this problem by putting most applications that enter a polling loop to sleep until there is keyboard input.

Some applications may appear to be in a polling loop when they actually are not. By default, the **pollsleep** feature might put these applications to sleep inappropriately, which would not adversely affect other processes, but would cause these applications to perform poorly. This situation is not common, but if you encounter it, you might want to disable the **pollsleep** feature.

To disable this feature, type at the DOS prompt:

```
C> merge set pollsleep off
```

drive Option

Some DOS applications (particularly applications designed to work in networks) detect whether a DOS drive is local or remote. These applications may not operate correctly if a DOS drive is local when the application expects it to be remote or vice versa.

To configure your DOS environment so DOS interprets ODT-DOS drives that refer to the shared DOS/UNIX file system (drives C:, D:, and J:, for example) to be local or remote, use the **merge** command with the following syntax:

merge set drive local|remote

The ODT-DOS default is **drive remote**. To make DOS interpret the shared DOS/UNIX file system as local, type:

```
C> merge set drive local
```

The **merge set drive local** command causes DOS applications that use DOS interrupt 21H system call function 44H (IOCTL) code 09H (Is Logical Device Local or Remote) to interpret ODT-DOS drives that refer to the shared DOS/UNIX file system (such as C:, D:, or J:) as local drives. The **merge set drive remote** command causes DOS applications using the same system call to interpret the same ODT-DOS drives as remote drives. Use these **merge** commands when necessary to configure your DOS environment so applications that are sensitive to local and remote drives work properly.

The following section describes the related **set handle** option.

handle Option

Some DOS applications (particularly applications designed to work in networks) detect whether a DOS file is local or remote. These applications may not operate correctly if a DOS file is local when the application expects it to be remote or vice versa.

To configure your DOS environment so DOS interprets file handles for files in the shared DOS/UNIX file system to be local or remote, use the **merge** command with the following syntax:

merge set handle local|remote

The ODT-DOS default is **handle remote**. To make DOS interpret files in the shared DOS/UNIX file system as local, type:

```
C> merge set handle local
```

The **merge set handle local** command causes DOS applications that use DOS interrupt 21H system call function 44H (IOCTL) code 0AH (Is Redirected Handle) to interpret files in the shared DOS/UNIX file system as local files. The **merge set handle remote** command causes DOS applications using the same system call to interpret files in the shared DOS/UNIX file system as remote files. Use these **merge** commands when necessary to configure your DOS environment so applications that are sensitive to local and remote files work properly.

display Option

The **merge display** command displays the current settings for **fastclk, pollsleep, drive,** or **handle**. The syntax for this command is:

merge display [*option*]

For example, to see the current setting for **fastclk**, type at the DOS prompt:

```
C> merge display fastclk
```

[1] displays the current setting for **fastclk**. To see the current settings for all **merge** command options, type:

```
C> merge display
```

return Option

The **merge return** option sets the DOS exit code to a numeric value (0 or 1) that corresponds to the state of the specified option. The syntax is:

merge return *option*

With this form of the **merge** command, the system assigns the value 1 to be "on" or "remote" and 0 to be "off" or "local." For example, if **fastclk** is "on" and you type:

```
C> merge return fastclk
```

the DOS exit code is set to the value "1". You can use the DOS batch file ERRORLEVEL command to check the **merge return** exit code and then take appropriate action. For example, you might include the following lines in a batch file to determine whether **fastclk** is on:

```
merge return fastclk
if errorlevel == 1 goto ISON
  echo FASTCLK is OFF
  goto END
:ISON
  echo FASTCLK is ON
:END
```

Using Batch Files For merge Commands

Instead of manually typing a **merge** command to set the value of **fastclk**, **pollsleep**, **drive**, or **handle** when you enter the DOS environment, you can include the appropriate command or commands in a batch file that is executed when you start DOS. In general, you should *not* include **merge** commands in your home directory AUTOEXEC.BAT file unless you want to run **merge** commands every time you run DOS. Instead, follow these procedures:

1. Create a batch file, in any convenient directory, that includes the **merge** commands you want to run. For example, you could create a file in your home directory called SPECIAL.BAT that includes the commands:

   ```
   merge set drive local
   merge set pollsleep off
   ```

2. When you want to run a DOS application that requires these **merge** commands, start the DOS environment with the command:

   ```
   $ dos +pfile
   ```

   where *file* is the name of the batch file. For example:

   ```
   $ dos +p/usr/wayne/special.bat
   ```

   The **merge** commands in SPECIAL.BAT are then effective for the duration of your DOS environment (or until you change them by issuing another **merge** command).

You can also tailor applications to run specific batch files automatically whenever you start them from the UNIX shell. Refer to Chapter 6 and to the description of the ±p option in Chapter 7 for further information.

Using AUTOEXEC.BAT And CONFIG.SYS Files

Because different users may want to include different commands in their AUTOEXEC.BAT or CONFIG.SYS files, ODT-DOS provides for both:

- System default AUTOEXEC.BAT and CONFIG.SYS files, which affect all users unless they explicitly specify otherwise.
- Personal AUTOEXEC.BAT and CONFIG.SYS files, which individual users can create to customize their own personal DOS environments.

System Default AUTOEXEC.BAT And CONFIG.SYS Files

The system default AUTOEXEC.BAT and CONFIG.SYS files, if they exist, are in the root directory. That is, their full path names are C:\AUTOEXEC.BAT and C:\CONFIG.SYS. Only CONFIG.SYS exists by default when you install ODT-DOS, but the system administrator can create AUTOEXEC.BAT at any time and include in it any commands that are useful for all users. The default CONFIG.SYS contains "DEVICE=" lines for the ODT-DOS expanded memory and mouse drivers.

If the root directory AUTOEXEC.BAT and CONFIG.SYS files exist, they are interpreted any time any user starts the DOS environment, unless the user has explicitly requested that these files not be interpreted.[1]

Personal AUTOEXEC.BAT And CONFIG.SYS Files

If you create a personal AUTOEXEC.BAT file in your home directory, ODT-DOS executes it whenever you start the DOS environment. ODT-DOS executes your home directory AUTOEXEC.BAT file after executing the root directory AUTOEXEC.BAT. You can also tell ODT-DOS to run *only* your personal AUTOEXEC.BAT, specify any other batch file that should be executed automatically when you run DOS, or tell ODT-DOS not to run any batch file when you run DOS. Refer to the description of the ±p option in Chapter 7 for further information.

ODT-DOS treats CONFIG.SYS files in a similar way. If you have a CONFIG.SYS file in your home directory,ODT-DOS interprets it after interpreting the root directory CONFIG.SYS every time you enter the DOS environment or start a DOS process. You can also tell ODT-DOS to interpret one or more configuration files instead of the default CONFIG.SYS files or not to interpret any configuration file. Refer to the description of the ±e option in Chapter 7 for further information.

[1] Note that ODT-DOS may behave differently when you start DOS commands or applications directly from the UNIX shell than it does when you start the DOS environment with the **dos** command. When you run a DOS command or application directly from the UNIX shell, ODT-DOS automatically runs AUTOEXEC.BAT only if you have used **dosadmin** or the **+p** option to configure the command or application to run AUTOEXEC.BAT. Refer to Chapter 6 and to the description of the ±p option in Chapter 7 for further information.

ODT-DOS Treatment Of CONFIG.SYS Commands

In general, ODT-DOS interprets CONFIG.SYS commands as you would expect. Following is additional information about how ODT-DOS treats each CONFIG.SYS command:

- **BREAK**: The BREAK command has the same effect on a ODT-DOS computer that it has on a conventional DOS personal computer. You can also use the ±x option, described in Chapter 7, to control how DOS interprets CTRL-BREAK. If you have a BREAK command in a CONFIG.SYS file that ODT-DOS interprets, it has priority over the ±x option.

- **BUFFERS**: The default number of buffers when you run DOS under ODT-DOS is 2. This value is defined in the DOS images at the time the images are created. You can override this value by including a BUFFERS command in any CONFIG.SYS file that ODT-DOS interprets when DOS starts. If there is more than one BUFFERS command in the CONFIG.SYS files that ODT-DOS interprets when DOS starts, the *highest* value is used. The BUFFERS command is effective only when you use actual DOS file systems, such as a DOS partition or the diskette drive. It is not useful when you use the shared DOS/UNIX file system.

- **COUNTRY**: The COUNTRY command works just as it does on a conventional DOS personal computer. It is effective only when you use actual DOS file systems.

- **DEVICE**: The DEVICE command works just as it does on a conventional DOS personal computer.

- **DRIVPARM**: The DRIVPARM command works just as it does on a conventional DOS personal computer. It is effective only when you use actual DOS file systems.

- **FCBS**: If there is more than one FCBS command in the CONFIG.SYS files that ODT-DOS interprets when DOS starts, the *highest* value is used. The FCBS command is effective only when you use an actual DOS file system.

- **FILES**: If there is more than one FILES command in the CONFIG.SYS files that ODT-DOS interprets when DOS starts, the *highest* value is used. The FILES command is effective only when you use an actual DOS file system. A DOS process can open a maximum of 123 files simultaneously in the shared DOS/UNIX file system.

- **LASTDRIVE**: The ODT-DOS default LASTDRIVE value is "N". This value is defined in the DOS images at the time the images are created. You can override this value by redefining LASTDRIVE in any CONFIG.SYS file that ODT-DOS interprets when DOS starts. If there is more than one LASTDRIVE command in the CONFIG.SYS files that ODT-DOS interprets when DOS starts, the *highest* value is used.

- **SHELL**: The SHELL command works just as it does on a conventional DOS personal computer.

- **STACKS**: ODT-DOS does *not* interpret any STACKS commands in any CONFIG.SYS files at DOS run time. The STACKS value is defined in the DOS images at the time they are created and cannot be changed unless you make new DOS images. The STACKS value used in the factory default DOS images is the same as the standard DOS default value: 9,128. That is, there are nine stacks with 128 bytes each. See *Administering ODT-DOS* for further information on changing STACKS and making new DOS images.

Chapter 2

Using DOS From The UNIX Shell

This chapter tells you more about the characteristics of ODT-DOS when you use it from the UNIX shell. It is organized as follows:

- **DOS Commands Unavailable from the UNIX Shell** lists DOS commands you can't use from the UNIX shell.
- **DOS Program Names** tells you about DOS naming conventions for executable files and how ODT-DOS recognizes DOS command names.
- **Configuring DOS Programs for Use from the UNIX Shell** explains how to make DOS programs executable from the UNIX shell.
- **Using DOS Drives from the UNIX Shell** describes techniques for using several specialized ODT-DOS disk drives.
- **Changing the UNIX Search Path** tells you how to add directories to the default ODT-DOS search path.
- **Using Special UNIX Shell Characters** describes in more detail than *Using ODT-DOS* how to use UNIX metacharacters in DOS commands.

DOS Commands Unavailable From The UNIX Shell

Several DOS commands have no known use from the UNIX shell. The UNIX system provides the same functions as some of these DOS commands, in many cases with commands having the same names. Other DOS commands (CHDIR, for example) are not useful from the UNIX shell because they affect a transient DOS environment that lasts only

as long as the command itself. The following DOS commands are not enabled for use from the UNIX shell:

APPEND	CLS	GRAPHICS	NLSFUNC	SHARE
ASSIGN	CTTY	JOIN	PROMPT	SUBST
BREAK	FASTOPEN	KEYB	PATH	VERIFY
CHCP	FDISK	MKDIR	RMDIR	
CHDIR (CD)	GRAFTABL	MODE	SET	

All DOS batch commands except FOR

Refer to "Configuring DOS Programs for Use from the UNIX Shell," later in this chapter, if you find that you need to use a DOS command that is not enabled for use from the UNIX prompt.

As in the DOS environment, the following additional restrictions apply to DOS commands:

- You can't use CHKDSK, FORMAT, or SYS on the shared DOS/UNIX file system. You can, however, use these commands on an actual DOS file system (such as a diskette drive or a DOS partition) as you would on any conventional DOS system.

- The DOS TIME and DATE commands can be used from the UNIX shell to display the time or date, but cannot be used to alter the ODT-DOS system clock. Note that the UNIX system also includes **time** and **date** commands, so if you use DOS TIME or DATE, refer to "Avoiding DOS/UNIX Program Name Conflicts" in *Using ODT-DOS*.

DOS Program Names

If you are a UNIX user who is unfamiliar with DOS, it can be useful for you to learn about the different kinds of names that apply to DOS executable programs and the DOS conventions for invoking programs. DOS programs have names that end in **.com**, **.exe**, or **.bat**. These endings are known as file name *extensions*. Each of these extensions indicates a particular type of executable DOS program. Although standard DOS requires programs to be named with one of these extensions, it allows you to run the programs without specifying the extension. For example, on a conventional DOS computer or using the DOS environment on the ODT-DOS system, you can run the **tree.com** command simply by typing:

```
C> tree
```

The UNIX shell does not interpret file names with extensions the same way DOS does. If you type tree at the UNIX prompt on a standard UNIX system, the shell looks for a file called **tree**, not **tree.com**.

ODT-DOS solves this problem by using links. A link is simply an alternative name for a file, an "alias" by which a file is known. For example, if the file **tree** is linked to the file **tree.com**, you can run **tree.com** either by typing tree.com or by typing tree. To make standard DOS commands executable from the UNIX shell, ODT-DOS links the DOS file names that include extensions to the same file names without extensions. The file **tree** is linked to **tree.com**, **comp** is linked to **comp.com**, and so on. Because these links exist, the UNIX shell can find the name when it searches for the command, and the DOS command is executed. The standard DOS commands and the corresponding linked UNIX files are in the directory **/usr/dbin**.

There is a second, smaller class of DOS programs that do not use file name extensions — the *built-in* or *internal* commands. These are often-used commands like COPY and DIR that are built into the DOS command interpreter, COMMAND.COM. When you type an internal command at the DOS prompt in a DOS environment, DOS interprets the command as it would on any conventional DOS personal computer. To allow DOS internal commands to be run from the UNIX shell, ODT-DOS uses files with names corresponding to each internal command (**dir** and **copy**, for example) that are stored in **/usr/dbin**. These files contain special codes that tell ODT-DOS to run DOS and execute the proper internal DOS command when you invoke a DOS internal command at your UNIX shell prompt.

Configuring DOS Programs For Use From The UNIX Shell

Administering ODT-DOS contains instructions on using **dosadmin** to configure DOS applications for use from the UNIX shell. It also summarizes equivalent manual procedures. You can use **dosadmin** or the equivalent manual procedures to configure standard DOS commands that aren't already configured for use from the UNIX shell. You can also use them to configure off-the-shelf or custom DOS applications you have installed on the ODT-DOS system fixed disk.

If you want to make a DOS internal command executable from the UNIX shell, you need to use a different procedure. The procedure is simple: just copy or link an existing file for an internal command in **/usr/dbin** to

the name of the command you want to enable. For example, if you want to make the DOS MKDIR command executable from the UNIX shell, you could type:

```
# cp /usr/dbin/dir /usr/dbin/mkdir
```

Make sure the new file has UNIX execute permission. You may also want to use the **dosopt** command to assign appropriate **dos** options to the new file. See Chapters 6 and 7 for more information on **dosopt** and **dos** options.

Using DOS Drives From The UNIX Shell

You can use DOS drives from the UNIX shell to access DOS files and programs stored outside the shared DOS/UNIX file system. To run a DOS command that operates on a file outside the shared DOS/UNIX file system, name the DOS drive containing the file in your command. For example, the command:

```
$ dir a:memos/july
```

displays a list of the files in the directory **memos/july** on the diskette in drive A:. To execute a DOS program stored outside the shared DOS/UNIX file system, precede the command with the word "dos" and specify the drive name as you would when using standard DOS:

```
$ dos a:cards
```

The same principles that apply to the physical diskette drive also apply to any real DOS file systems, including the physical DOS partition, virtual DOS partitions, and virtual floppies. The following examples illustrate these principles:

- Run a program called TRICKS stored on the physical DOS partition:

  ```
  $ dos e:tricks
  ```

- Run a DOS program called SHOWDATA stored on the virtual floppy **/usr/paul/vflop** to display information about the file **messages** in your home directory (drive D:):

  ```
  $ dos +ab:=/usr/paul/vflop b:showdata d:messages
  ```

- Display the contents of the virtual partition /usr/sam/vpart:

  ```
  $ dos +af:=/usr/sam/vpart dir f:
  ```

Changing The UNIX Search Path

Use standard UNIX procedures if you want to change your search path to include directories containing DOS programs. The default search path for all users is commonly defined in the file **/etc/default/login**. Users who want to modify the default path typically define their own paths in their home directory **.profile** files.

Do not include a DOS drive designation such as C: in your path definition. Only directories in the shared DOS/UNIX file system can be included in the UNIX search path. ODT-DOS automatically translates the UNIX path into a DOS path specifying drive C: when you enter the DOS environment.

Refer to Chapter 5 for additional techniques for managing your search path.

Using Special UNIX Shell Characters

There are several special characters that are interpreted by the UNIX shell when they are used in a UNIX command line. These are called *metacharacters* and include the following:

```
< > * ? | & $ ; \ " ` ' ^ ( ) [ ] #
```

For information on using these characters with the UNIX shell, refer to your UNIX documentation on the shell.

When you use any of these characters on a command line with a DOS command, ODT-DOS interprets them in the standard UNIX way. You can therefore use them as necessary with DOS commands just as you would use them with UNIX commands. If you are used to using any of these metacharacters on a conventional DOS system (or in the ODT-DOS environment), however, you should be aware that UNIX treats them differently than DOS does. For example, if you issue the DOS command:

```
C> copy *.com a:
```

in the DOS environment, DOS copies all files ending with **.com** in your current directory on drive C: to your current directory on drive A:. The equivalent command from the UNIX shell is:[1]

```
$ dos copy *.com a:
```

[1] Note that the command must start with dos to avoid a conflict with the UNIX copy command.

This command works differently than it does in the DOS environment because the UNIX shell translates the asterisk (*) before the DOS COPY program receives the command. By the time the command is passed to DOS, it has been translated into something like the following:

```
$ dos copy file1.com file2.com file3.com a:
```

An error message results since DOS COPY accepts only one source file name. Similar errors can occur with other UNIX shell metacharacters.

You can prevent the UNIX shell from interpreting shell metacharacters by using the standard UNIX shell escape character, the backslash (\). For example:

```
$ dos copy \*.com a:
```

The backslash prevents the UNIX shell from interpreting the metacharacter "*," and the command is passed to DOS and interpreted correctly.

You can also use both the single quote (') and double quote (") symbols to prevent the UNIX shell from interpreting metacharacters. The quotes in the command:

```
$ dos copy "*.com" a:
```

tell the UNIX shell not to interpret the metacharacter asterisk (*), and the command is passed to DOS in the desired form. Single quotes work equally well.

Some DOS commands require the quote symbol as part of their command syntax. The quote must be passed literally to DOS rather than being used by the UNIX shell to prevent interpretation of other metacharacters. The DOS FIND command, for example, uses quotes to surround a character string that is being searched for. A typical use of this command in the DOS environment would be:

```
C> find "October" memo
```

(This command displays all lines in the file **memo** containing the word "October".) If you issue the command from the UNIX shell in this form, the UNIX shell strips away the quotes and passes the command in an illegal form to DOS. To avoid this unwanted behavior, type:[1]

```
$ dos find \"October\" memo
```

You can sometimes avoid unwanted interpretation of metacharacters by including DOS drive designations when you run a command. To accomplish the copying operation described above, for example, you can type:

```
$ dos copy c:*.com a:
```

Provided you do not have a file in your current directory that matches the character string "c:*.com" (including the "c:"), this command works as expected. When you issue this command, the UNIX shell first looks for files matching the character string "c:*com" in order to carry out the proper substitution for the metacharacter asterisk (*). Because the shell cannot find matching files, interpretation of the metacharacter does not occur. The string "c:*.com" is passed literally to DOS, which interprets the drive designation and metacharacter as expected.

[1] Again note that the command starts with dos to avoid a conflict with the UNIX **find** command.

Chapter 3

Advanced Printing

The procedures for printing with DOS described in *Using ODT-DOS* are sufficient for most purposes. Using those procedures, you can print files using commands such as print *filename* or copy *filename prn*, print the contents of your screen with the PRT SC key, and print from DOS applications.

You may want to customize the operation of DOS print commands, use more than one printer, or circumvent the UNIX print spooler so you can observe each line of DOS printer output while the DOS print command is in progress. This chapter explains how to accomplish these tasks in the following sections:

- **Selecting and Customizing Print Streams** explains how to use the ODT-DOS **printer** command to direct DOS printer output to different printers.
- **Changing the Printer Timeout** describes the procedures to use when the default printer timeout is inconvenient for your printing applications.
- **Attaching Printers Directly to DOS Processes** explains how to avoid using the UNIX print spooler so DOS has complete control of the printer.
- **Using the DOS MODE Command** contains a hint concerning the MODE command.

Refer to *Administering ODT-DOS* for further information on administering DOS printers.

Selecting And Customizing Print Streams

By default, ODT-DOS redirects all DOS printer output sent to LPT1, LPT2, LPT3, or PRN to the UNIX print spooler for printing on a printer named **doslp**. When you use printing functions as described in *Using ODT-DOS*, you don't need to be aware that ODT-DOS uses the UNIX spooler or that the printer you use is named **doslp**. The printer is shared in a convenient way between DOS and UNIX, and DOS print jobs are queued and printed along with UNIX print jobs.

The system administrator may configure additional UNIX printers, with different names, so they can process DOS printer output. To use any available UNIX printer for DOS printing, use the ODT-DOS **printer** command to correlate a print stream with a particular UNIX printer and print command. The syntax for selecting a print stream, a printer, and a UNIX print command is:

printer [*print_stream*] **unix** "*print_command*"

where *print_stream* is LPT1, LPT2, or LPT3 and *print_command* is a UNIX command that processes the specified print stream. If you do not specify a print stream, **printer** assumes LPT1 by default. Use the **printer** command in the DOS environment. Assume, for example, that you want to direct DOS printer output sent to LPT2 to a UNIX printer named "laser." Follow these steps:

1. Start a DOS environment if you haven't already started one.

2. Use the following **printer** command to direct print stream LPT2 to to the printer named "laser":

   ```
   C> printer lpt2 unix "lp -dlaser"
   ```

   In this command, the **-d** ("destination") option to the **lp** command identifies the printer named "laser."

3. To send DOS printer output to the printer, name the DOS print stream in your DOS print command. For example:

   ```
   C> copy letter.txt lpt2
   ```

You can direct printer output from DOS applications to any UNIX printer using the same procedures.

If you want a **printer** command to be effective every time you use the DOS environment, you can include it in an AUTOEXEC.BAT file.

Note that the *print_command* that you specify when you use the **printer** command is typically lp -d*printer*, where *printer* is the name of a UNIX printer. However, *print_command* can be *any* UNIX command that you choose to use to process a print stream.

Changing The Printer Timeout

When you use a DOS application that prints, ODT-DOS by default sends DOS printer output to the UNIX spooler when either of the following conditions is true:

- You exit the application and return to your DOS or UNIX prompt.
- More than 15 seconds have elapsed since the application has sent a character to be printed.

This default is convenient for most applications. However, under certain conditions you may want to change the printer timeout from 15 seconds to a different value. For example, if your application pauses for more than 15 seconds while printing, the printer output sent before and after each pause is spooled as a separate UNIX print job.

To change the printer timeout in the DOS environment, use the ODT-DOS **printer** command with the syntax:

printer [lpt*n***] unix /t[***timeout***]**

In this command, LPT*n* identifies the print stream for which you wish to change the timeout. LPT1 is assumed if you do not specify a print stream. *timeout* is a value, in seconds, between 5 and 3600. For example, if you want to increase the timeout for print stream LPT2 to one minute, type the command:

```
C> printer lpt2 unix /t60
```

Using the /T option without specifying a value returns the printer timeout for the specified print stream to the default of 15 seconds.

The timeout value "0" has a special meaning: it delays the spooling of DOS printer output until you exit the application.

You can use the ±s option instead of the PRINTER command to control the printer timeout. Refer to Chapter 7 for information on the ±s option.

Attaching Printers Directly To DOS Processes

In some situations, you might want to avoid the UNIX spooler when using DOS printing. When you use the UNIX spooler, each DOS print job is sent to the printer as a complete unit. You therefore have no opportunity to examine any printed output before the entire job is sent to the printer. If you want to view DOS printer output while the DOS printing application is still running, you cannot use the UNIX spooler.

Attaching a printer directly to a DOS process allows you to see DOS printer output as it happens. When a printer is directly attached to a DOS process, only the user running that DOS process has access to the printer. If the printer is normally used with the UNIX spooler, other users may continue to submit print jobs intended for that printer. However, these print jobs are not printed until the DOS process exits and the printer is enabled again for UNIX use.

If the printer is currently set up for UNIX printing, the system administrator should use the **disable** command to disable UNIX printing on that printer before any user directly attaches it to DOS. When the DOS printing has been completed, the system administrator can reenable UNIX printing with the **enable** command. Refer to *Administering ODT-DOS* and *UNIX System Administrator's Guide* for more complete information on disabling and enabling UNIX printing and other printer administration procedures.

Follow these steps to directly attach a printer to a DOS process:

1. Start DOS using the **+a** option to specify the physical printing device you want to use. ODT-DOS uses the device names **lp0**, **lp1**, and **lp2** to identify the first, second, and third parallel printer ports. Use the name that corresponds to the port your printer is attached to. For example, if your printer is attached to **/dev/lp0**, you can start the DOS environment with the command:

   ```
   $ dos +alp0
   ```

   Consult the manuals for your computer and the ODT-DOS **/etc/dosdev** file if you are uncertain how to identify your printer port.

2. You can now use any of the standard DOS printing techniques, including the PRINT command, copying a file to the printing device, and printing from an application. The name of the printing device (LPT1, LPT2, or LPT3) is determined by DOS and does not necessarily correspond to the name of the physical port you attached in the previous step (**lp0**, **lp1**, or **lp2**). DOS checks each of the directly attached physical printer ports, in order, to identify the first port connected to a printer. No matter which physical port the printer is attached to, DOS names the first printer it finds "LPT1," the second printer it finds "LPT2," and so on. If you have only one printer and it is attached to the physical port named **lp2**, DOS views this printer as LPT1.

   For example, if you have directly attached one printer, you can print a file with the command:

   ```
   C> copy filename lpt1
   ```

   When your DOS session ends, the printer can be enabled again for UNIX printing.

NOTE: When you directly attach a printer to a DOS process as described in this section, you can use all DOS PRINT command options when you use that printer.

Using The DOS MODE Command

If you use the MODE command to configure a DOS printer, the command has no effect unless the printer is directly attached to the DOS process.

Chapter 4

Running UNIX Programs From DOS

The ODT-DOS **on** utilities allow you to run UNIX programs from the DOS environment and view the output as though the programs were actually running under DOS. You can also view status information concerning UNIX programs and control their execution and output from the DOS environment.

The **on** utilities can only be used to execute *noninteractive* UNIX commands — those that do not initiate a conversation with the user. You must switch to a UNIX screen (by forking a UNIX shell, for example) or quit the DOS environment if you want to run interactive UNIX programs such as the **vi** text editor.

With the **on** utilities you can:

- Execute UNIX commands without switching to a UNIX screen or exiting the DOS environment.
- Extend the functionality of the DOS environment by executing UNIX commands on DOS files as if they were DOS commands.
- Run UNIX programs in the background and view their output at a later time.

The **on** utilities include three commands that run under DOS: **on**, **jobs**, and **kill**.

This chapter is organized as follows:

- **Specifying the Machine and UNIX Command** explains how to use the different forms of the **on** command.
- **Search Path and Other Environment Considerations** tells you about the environment that **on** uses, how DOS finds the **on** command, and how UNIX finds your specified UNIX command.
- **Breaking Out Of on** shows you how to abort an **on** task or switch it from the foreground to the background.

- **Running on Jobs in the Background** explains the **on** job table and several techniques for managing background **on** tasks.
- **Using Pipes and Redirection** illustrates the proper use of DOS and UNIX pipe and redirection mechanisms in **on** tasks.
- **Summary of Restrictions and Cautions** lists precautions and limitations that apply to **on** tasks.

Specifying The Machine And UNIX Command

There are two major command forms for **on**. The syntax for the first form is:

on unix|- *unixcommand* [&]

In this syntax, **unix** or the placeholder "-" specifies the machine or operating system on which *unixcommand* should run. Both **unix** and the placeholder "-" refer to the local UNIX system.

For example, type:

```
C> on unix cal
```

or

```
C> on - cal
```

In either case, the UNIX **cal** command runs and displays a calendar of the current month on your screen.

UNIX commands can contain all options and arguments exactly as they would be typed at a UNIX prompt. For example:

```
C> on unix pr -o10 -w65 -l54 -d /tmp/longfilename1
```

Specify UNIX file names with their full UNIX names, not with their mapped DOS names.

The **on** command cannot execute multiple UNIX commands separated by semicolons unless the commands are surrounded by parentheses. For example:

```
C> on unix (ls ; cat names)
```

The **on** command automatically converts the text output of the UNIX command from UNIX format to DOS format. That is, the ODT-DOS **unix2dos** utility is built in.

If **on** cannot execute the requested UNIX command, either because it cannot find a requested file or because you do not have execute permission for a requested file, then **on** returns the following error message:

unixcommand: access denied or file not found

where *unixcommand* is the name of the command that **on** attempted to run.

The on Command And DOS Drives

In many circumstances, the two commands:

```
C> on unix unixcommand
C> on - unixcommand
```

produce identical results. These commands operate differently in the following respects:

- The command:

  ```
  C> on unix unixcommand
  ```

  runs the specified UNIX process in your current directory on drive C: *even if your current drive is not drive C:*.

- The command:

  ```
  C> on - unixcommand
  ```

  runs the specified UNIX process in the current directory of your current drive, provided it is a drive (such as C:, D:, or J:) that accesses the shared DOS/UNIX file system.

When you use the DOS environment, DOS keeps track of your current directory on drive C: even when you run commands on other drives. This

is the directory where UNIX commands specified with **on unix** execute. For example, if your current working drive is drive C: and you type:

```
C> on unix rm temp
```

on removes the file **temp** in your current working directory on drive C:. If your current working drive is drive A: and you type:

```
A> on unix rm temp
```

on removes the *same* file as in the previous example — the file **temp** in your current working directory on drive C:.

When you use the form **on** - *unixcommand,* **on** executes your specified UNIX command on your current drive if it is a drive (such as C:, D:, or J:) that accesses the shared DOS/UNIX file system. If your current drive is not one of these drives, **on** fails and displays an error message.

Using UNIX Command Names Directly

The second major form of the **on** command allows you to run UNIX commands without typing on unix for every command. When you want to avoid typing on unix, copy or link the **on** program to the names of the UNIX commands you want to run directly from the DOS prompt.

Copying on

The executable DOS file that contains the **on** command is \USR\DBIN\ON.EXE. To make a UNIX command executable directly from the DOS prompt, you can copy ON.EXE to a file with the name of the UNIX command you want to run. Include the file name extension .EXE in the renamed copy of ON.EXE.

Assume, for example, that you have a UNIX program called **getname** that displays a user's full name when given either a first or last name. To make **getname** executable under DOS, copy ON.EXE with the command:

```
C> copy \usr\dbin\on.exe getname.exe
```

You could then type:

```
C> getname joe
```

from the DOS prompt, and Joe's full name is printed on your DOS screen.

This form of the **on** command runs like the **on** - *unixcommand* form. That is, the specified UNIX command runs in the current directory of the current drive, provided it is a drive that accesses the shared DOS/UNIX file system. If it is not one of these drives, **on** fails and displays an error message.

All options and arguments are entered following the renamed copy of **on** exactly as they would be at the UNIX prompt. This feature allows you to make copies of **on** with the names of UNIX utilities and use them as if they were DOS utilities.

You can make copies of ON.EXE in any directory on any drive.

Observe the following precautions and restrictions:

- The file names of your UNIX commands must have the extension .EXE, just like ON.EXE does.
- As with any DOS command in the DOS environment, the copies of ON.EXE must be in your DOS search path or your current directory.
- The UNIX command that ON.EXE is renamed to resemble must contain only lowercase letters in its name. (Other commands can still be run by preceding the command with on unix.)
- The name of the UNIX command cannot violate any of the DOS naming conventions. For example, you could not create an **on** version of a UNIX shell script named **mycalendar** because its name contains ten letters. (You can still run **mycalendar** by using the command on unix mycalendar, however.)
- Do not try to make copies of ON.EXE with the names of UNIX programs called **jobs** or **kill**. The **on** utilities interpret **jobs** and **kill** as built-in commands. Whenever you run these commands directly from your DOS prompt, ODT-DOS runs the **on** utilities versions. If UNIX versions of these commands exist and you want to execute them from the DOS prompt, use the first form of the **on** command, like this:

  ```
  C> on unix jobs
  C> on unix kill pid
  ```

- Do not make copies of **on.exe** with the names of DOS internal commands. The DOS shell interprets DOS internal commands, such as TYPE, as soon as they are entered. You therefore cannot run **on** utilities versions of these commands directly from the DOS prompt. To run a UNIX command with the same name as a DOS internal command, use the first form of the **on** command, like this:

  ```
  C> on unix type cat
  ```

Linking on ln The Shared DOS/UNIX File System

The **on** command is stored in \USR\DBIN\ON.EXE in the shared DOS/UNIX file system. Rather than using up disk space by making copies, you should, whenever possible, use the UNIX **ln** command to create links to the **on** program. The advantage of using the UNIX ln command instead of the DOS COPY command is that **ln** allows a single file to have more than one name without taking up extra disk space.

When ODT-DOS is installed, several UNIX programs are already linked to \USR\DBIN\ON.EXE. These programs are:

cat	**df**	**egrep**	**lp**	**spell**
chmod	**diff**	**fgrep**	**ls**	**tail**
cmp	**dosopt**	**grep**	**mv**	**wc**
cp	**du**	**ln**	**pr**	

Like ON.EXE, these links are in the directory \USR\DBIN. Because these links exist, you can use any of these commands in the DOS environment just as though they were DOS commands.

Suppose, for example, you want to search DOS text files for particular character strings and display the ends of DOS text files. You can accomplish these tasks by running the UNIX **grep** and **tail** commands at your DOS prompt:

```
C> grep -n Wilson letter.txt
C> tail -23 letter.txt
```

These commands print every line in LETTER.TXT that contains the word "Wilson" and then display the last 23 lines of the same file on your screen.

If you want to add another UNIX program to the list of linked commands, for example the **cal** program, you can type:

```
C> ln /usr/dbin/on.exe /usr/dbin/cal.exe
```

The precautions and restrictions that apply to copies of ON.EXE also apply to links to ON.EXE. In addition, you cannot link ON.EXE to a file in another file system. See your UNIX manuals for further information on the **ln** command.

Search Path And Other Environment Considerations

All forms of the **on** command use both DOS and UNIX search paths. If you use the form **on unix** *unixcommand* or **on** - *unixcommand*, DOS must be able to find the file ON.EXE. ON.EXE is in \USR\DBIN, which is in the default DOS search path. If you use the form *linked_unix_command* or *copied_unix_command*, DOS must be able to find the file that is linked to or a copy of ON.EXE. With any form of the **on** command, UNIX must be able to find the UNIX command named in the **on** command line.

ODT-DOS executes UNIX commands that you run with **on** under the standard Bourne shell, **sh**. Any UNIX environment variables exported by the shell that started your DOS environment are available to UNIX programs executed with **on**.

Breaking Out Of on

Unless **on** is running as a detached task, as described in the next section, you can interrupt it by pressing CTRL-C or CTRL-BREAK. The following prompt is displayed:

```
a - abort, c - continue, d - detach:
```

Entering a kills the job, clears the job from the job table, and returns you to a DOS prompt. Entering c allows the process to continue as before in the DOS foreground. Entering d detaches the UNIX process from your terminal, returns your DOS prompt, and continues to run the specified UNIX program in the background under UNIX. The next section describes the job table and detached tasks.

Running on Jobs In The Background

The **on** utilities allow you to:

- Start one or more UNIX commands from your DOS prompt and run them in the background.
- Detach an **on** command that is currently running in the foreground so that it continues to run in the background. Your DOS prompt returns and you can issue other commands while your **on** program continues.
- View a table showing the status of all currently executing and completed **on** commands.
- View the output of detached commands, including commands that have finished executing, at any time.

The following sections describe the features for controlling jobs executed with **on**.

Using The Job Table

ODT-DOS maintains a job table that keeps track of detached UNIX processes initiated by **on**—that is, processes that are placed in the background of the UNIX environment. The job table shows the status of up to ten detached **on** commands. Each DOS environment has its own job table.

The job table holds one entry per **on** command. Each UNIX command, however, may create several subordinate processes. For example a single **spell** command takes only one space in the DOS job table, but appears in a UNIX **ps** (process status) list as four or five processes.

All UNIX systems limit the number of simultaneous processes each user can run. This limit is typically 20 processes per user. If several UNIX commands initiated with **on** each create several processes, the limit for processes per user may be reached before the limit for entries in the job table.

Two Ways To Detach Tasks

You can detach UNIX tasks initiated with **on** so they run in the UNIX background in either of two ways:

1. Add an ampersand (&) to the end of the **on** command. For example:[1]

   ```
   C> spell memo &
   ```

   Note that the ampersand must be preceded by a space. The following command is incorrect:

   ```
   C> spell memo&          (incorrect)
   ```

2. Initiate an **on** command and later interrupt it with CTRL-C or CTRL-BREAK. For example:

   ```
   C> spell memo
   CTRL-C
   ```

   The following prompt is displayed:

   ```
   a - abort, c - continue, d - detach
   ```

   When you enter d, the task is placed in the background under UNIX.

In either case, **on** responds with a message in the form:

[jobnumber] pid

where *jobnumber* is the job number of the task in the DOS job table, and *pid* is the UNIX process ID number returned by UNIX. Your DOS prompt then returns and you can issue additional commands (including **on** commands) while your detached process continues executing in the background.

Any output produced by a detached **on** task (for example, the spelling errors found by the **spell** command shown previously) is by default stored in a temporary file. You can see the contents of this temporary file by reattaching the task as described later under "Reattaching to Detached Tasks." You can also use pipes and redirection to send the output of detached tasks to any file or program you choose. See "Using Pipes and Redirection," later in this chapter.

Keep these factors in mind when initiating detached tasks:

- If the job table is full at the time you attempt to start another process, **on** returns the error message:

  unixcommand: **job table full**

[1] This example and all subsequent examples of standard UNIX commands in this chapter use the second on command form. That is, the words on unix are not explicitly included in the command since the names of these UNIX commands are linked to \USR\DBIN\ON.EXE. All of these examples work equally well if you include on unix (or on -) at the beginning of each command.

where *unixcommand* is the name of the UNIX process that **on** attempted to run. This error message is returned when every position in the job table is filled, whether with "Done" entries or "Running" entries.

- If the maximum number of UNIX processes has been reached when you issue an **on** command, **on** returns this error message:

```
Unix exec failed
```

This message may occur when there are still places to fill in the job table. The total number of UNIX processes you can start is set by UNIX independently of the maximum number of job table entries available.

Keeping Track Of Detached Tasks

You can use the **jobs** command to perform any of the following tasks:

- Display job table information.
- Clear the job table of entries for completed jobs.
- Reattach to detached jobs.

Invoking **jobs** with no arguments displays the current job table in the following format:

```
JOB   STATE      EXIT STATUS     COMMAND
[1]   Running                    unixcommand1
[2]   Done       exit(0)         unixcommand2
```

If the job table is currently clear, invoking **jobs** returns you to the DOS prompt. The job table is clear when no detached jobs have yet been run or when you have cleared the job table of all entries. The four columns in the job table report the following information:

1. The "JOB" column shows the job ID number that **on** assigned to each process when the process was detached.
2. The "STATE" column indicates whether the process is running or done.
3. The "COMMAND" column shows the UNIX command that was requested.

4. The "EXIT STATUS" column shows one of the following values:

exit(*nn*)	The job terminated with an exit status of *nn*. An exit status of 0 usually means the process terminated normally. Any other value may indicate an abnormal termination of the process.
unknown	The job terminated, but **on** was unable to determine its exit status. This condition does not normally occur.
signal(*nn*)	A signal was received that killed the process. In this case, *nn* indicates the signal received.
coredump(*nn*)	The signal received caused a core dump to occur. This is a special case of **signal**.
err3(*nn*)	An error in the functioning of **on** or ODT-DOS has occurred.

Completed processes remain in the job table until you clear them by entering **jobs** with a single hyphen as an argument. When you enter:

```
C> jobs -
```

these events occur:

- The current status of the job table is displayed, including all currently "Done" entries.
- The "Done" entries are cleared from the job table.
- Any temporary files in the \TMP directory that were associated with the "Done" jobs are removed.

Once jobs are cleared from the job table, they can no longer be reattached. Output from any task being saved in temporary files for reattachment and review is discarded.

Reattaching To Detached Tasks

Using the **on** utilities, you can reattach a task that has been detached. Reattaching allows you to:

- View all previous output of a currently running program and continue to view new output as the program sends it to your screen.
- View the output of a program that has finished running.

To reattach either to a currently running job or to a completed job, use the **jobs** command in the form:

```
jobs %jobnumber
```

where *jobnumber* is the job number of the detached task.

For example, suppose you start the UNIX **spell** program as a detached task by typing:

```
C> spell memo &
```

When you type this command, **on** displays a job number and process ID similar to:

```
[1] 4376
```

To reattach to your **spell** job, type the command:

```
C> jobs %1
```

If you decide to reattach to the detached job and don't know the job number, type the **jobs** command to find out. A display similar to this is printed:

```
JOB   STATE      EXIT STATUS     COMMAND
[1]   Running                    spell memo
[2]   Done       exit(0)         who
```

Using the percent sign without a number reattaches you to the lowest numbered task in the job table:

```
C> jobs %
```

When you reattach to a currently running job (that is, one that appears as "Running" in the job table), you see all the output produced by the job up to the time you reattach to it. The job then continues to run, and you see any additional output as it is printed. If you wish, you can detach the task again at any time while it is running. Each time you reattach it, you see *all* output printed up to the time you reattach.

When you reattach to a completed job (one that appears as "Done" in the job table), you see all output printed by the job. The temporary file storing the output is removed and the job table entry for that job is cleared. The temporary file is also removed and the job table entry cleared if the job completes while it is attached.

Saving Output From Completed Jobs

If you reattach to a completed job as described in the previous section, you can see the output of your completed job, but the output is not saved for future reference. One way to save output of detached jobs is to use DOS redirection at the time you reattach a job. For example, assume a **spell** job has completed and appears as follows in the job table:

```
JOB   STATE      EXIT STATUS     COMMAND
[1]   Done       exit(0)         spell memo
```

You can save the list of spelling errors when you reattach to the job by typing the command:

```
C> jobs %1 > typos
```

This command redirects the list from your screen into a file called **typos**.

You can also use pipes and redirection when you first issue the **on** command. See "Using Pipes and Redirection," later in this chapter.

Stopping Detached Jobs

You can halt detached jobs and clear them from the job table by reattaching, pressing CTRL-C or CTRL-BREAK, then responding a (for "abort") to the resulting prompt.

You can also stop a UNIX process by using the **on** utilities **kill** command, which runs under DOS, in this form:

kill [*-signal*] *%jobnumber* [. . .]

where *jobnumber* is the task number of the job in the job table and *signal* is the UNIX signal to be sent to the process.

The **on** utilities **kill** command works exactly like the UNIX **kill** command except that it accepts *%jobnumber* instead of the process ID number argument. See the descriptions of *kill* and *signal* in your UNIX documentation for more information. The default signal sent is 15. A signal of 9 may be used for a sure kill. Note, however, that entering **kill -9** is bad practice unless other **kill** commands have failed, because the UNIX program has no chance to perform cleanup operations before exiting.

Using Pipes And Redirection

With the **on** command, you can use either DOS or UNIX pipes or redirection, or even combine the DOS and UNIX versions of these mechanisms in a single command. You can accomplish nearly all useful operations using the DOS mechanisms. The following sections therefore concentrate on DOS pipes and redirection. A few comments on UNIX pipes and redirection are included in "Using Pipes and Redirection in Detached Tasks," later.

Using DOS Pipes

You can use DOS pipes (|) with UNIX programs that you invoke using **on** just like you use them with DOS programs. The following examples show two different ways to accomplish the same task:

```
C> ls | find "chap"
C> ls | grep chap
```

Both versions locate and display all the file names in the current directory that contain the characters "chap." In the first example, output from the UNIX **ls** command is returned to the DOS environment, and the DOS pipe routes it through the DOS FIND program. In the second example, **on** returns the **ls** output to the DOS environment, and a DOS pipe rechannels it to the UNIX **grep** command.

Redirecting DOS Output

You can redirect the output of **on**-initiated commands to specified files. DOS output redirection (>) allows you to direct the output to a file on any disk drive or directory where you have write permission. For example, you could put the results of a command onto a diskette:

```
C> ls | grep chap > a:tempfile
```

DOS redirection of the final output of an **on** command works because **on** sends the output back to your DOS environment. Thus, you can redirect the output of both UNIX and DOS commands with the DOS output redirection mechanism (>).

Redirecting DOS Input

You can use the DOS input redirection mechanism (<) to redirect standard input from a file or from the DOS keyboard to an **on** command.

Input redirection from a file can be useful when you want to run a UNIX command that operates on a file contained on a DOS device, such as drive A:. The following command, for example, does not work because **a:names** is not a meaningful UNIX file name:

```
C> spell a:names          (incorrect)
```

Because the UNIX **spell** command can read from the standard input as well as from a file, you can use input redirection in a command like this to accomplish the operation:

```
C> spell < a:names
```

The DOS redirection mechanism "<" works together with the DOS drive designation "a:" to transfer the contents of the file **names** to the UNIX **spell** command. You cannot use this form of redirection with UNIX programs that cannot read from the standard input. For example, the UNIX **ls** (directory listing) program cannot read from standard input, so the following command does not work:


```
C> ls < a:                (incorrect)
```

You can also redirect the input for a UNIX command from the DOS keyboard. For example, you can direct keyboard input into the UNIX **mail** command as follows:

```
C> on unix mail jim < con:
This is a mail message created in the DOS environment
and sent with the UNIX "mail" command.
CTRL-Z
```

CTRL-Z closes the DOS standard input, which also effectively closes the UNIX standard input.

Using Pipes And Redirection In Detached Tasks

A detached **on** command that includes DOS pipes or redirection may not produce the results you want. This can happen because the DOS command processor, COMMAND.COM, interprets the pipe and redirection symbols (<, >, and |), while **on** interprets other parts of the command line, including the ampersand (&).

For example, if you want to redirect the output of the UNIX **spell** command into a file called TYPOS, you might incorrectly issue the following command:

```
C> spell memo > typos &   (incorrect)
```

This command, however, does not put the list of spelling errors in the file TYPOS. Instead, it puts the job number and process ID of the **spell** command (which **on** returns to the DOS environment) in TYPOS.

To save output produced by **spell** you can do either of the following:

- Issue the command in the form:

```
C> spell memo &
```

  and then redirect the output when you reattach, as described previously under "Saving Output from Completed Jobs."

- Use UNIX output redirection rather than DOS output redirection when you issue the **spell** command.[1]

The **on** utilities use the special characters "{" to mean UNIX input redirection, "}" to mean UNIX output redirection, and "!" to mean a UNIX pipe. When you use these characters in an **on** command, UNIX does the redirection or piping operation, and the results are often easier to understand. To use UNIX output redirection in the **spell** example illustrated above, type:

```
C> spell memo } typos &
```

This example displays a job number and process ID on your screen when you issue the command. Your DOS prompt returns, as expected, and the task runs in the background. When the **spell** program completes, the results are in the file TYPOS.

When you use UNIX redirection as illustrated in the previous example, the output of the **on** command is not returned to the DOS environment and is not converted into DOS text format. You can use the ODT-DOS **unix2dos** command to convert the file to DOS text format.

The following example shows how UNIX pipes and redirection can be useful. It shows both incorrect and correct ways of creating a file called NAMES containing a sorted list of users currently logged into the ODT-DOS system.

---

[1] You can also use quotes to prevent the pipe and redirection symbols from being interpreted by DOS. For example:

```
C> spell "memo > typos" &
```

This command, however, leaves the contents of the file TYPOS in UNIX text format.

```
C> on unix who | on unix sort > names &      (incorrect)
C> on unix who ! sort ! unix2dos } names &  (correct)
```

If you issue the first command, it is not detached until the **who** program finishes, and the file NAMES contains a job number and process ID rather than a sorted **who** list. This is probably not the result you want. The second command uses a UNIX pipe to send the list output by the **who** program directly to the UNIX **sort** program, without returning to DOS or using a second **on** command. Because the output is not returned to the DOS environment, it remains in UNIX text format rather than being converted to DOS text format. The **unix2dos** program is therefore used to convert the list to DOS format, following which UNIX output redirection (}) puts the list into the file NAMES.

To be properly interpreted by **on**, the special redirection and pipe characters ({, }, and !) require spaces surrounding them. The following examples, which omit the required spaces, are incorrect:

```
C> spell memos}typos &          (incorrect)
C> on unix who!sort!unix2dos}names & (incorrect)
```

4

Summary Of Restrictions And Cautions

- You cannot use the **on** command to run interactive UNIX programs.
- Since **on** runs UNIX programs under the Bourne shell, any UNIX commands or program you run with **on** must be executable under the Bourne shell.
- You cannot initiate multiple UNIX commands with a single **on** command containing semicolons unless the UNIX commands are surrounded by parentheses.
- When the **jobs** command reports the status of a job as "Done," it means the job has completed, but not necessarily successfully. Jobs that terminate for any reason are identified as "Done" in the job table.
- Copies of or links to **on** cannot be named after UNIX programs that contain an uppercase letter in their names. However, such programs can still be run by preceding the command name with on unix.

- Since the **on** command interprets the characters !, {, and } as special pipe and redirection symbols, you cannot use these characters in file names unless you precede them with a backslash. For example, if you have a UNIX-executable file named **{map}**, you would invoke it via **on** with the command:

```
C> on unix \{map\}
```

- The number of simultaneous **on** jobs you can execute is limited by the size of the job table, which can accommodate 10 jobs, and by the number of simultaneous processes allowed by UNIX for each user, which is typically 20. The UNIX limit includes jobs started with **on** as well as any other simultaneously executing jobs started from the UNIX shell. If you have started several UNIX programs from the UNIX shell, you may reach the UNIX system limit before reaching the number of **on** jobs accommodated by the job table. Also note that a single **on** command (such as **on unix spell**) can cause multiple UNIX processes to execute, and each process is counted in the UNIX system limit.

- If you detach a command containing DOS pipes or redirection, the results may not be what you expect. See "Using Pipes and Redirection in Detached Tasks," earlier.

- The **on** utilities are not tested with and may not run under DOS shell processors that replace COMMAND.COM.

Chapter 5

Other Advanced Topics

This chapter provides information of interest both to advanced ODT-DOS users and to the ODT-DOS system administrator. It covers a wide variety of techniques for customizing DOS and improving ODT-DOS efficiency. It is organized as follows:

- **Running DOS in an X Window** tells you how to use ODT-DOS in an X Window System environment.
- **Using Expanded Memory (EMS)** describes the CONFIG.SYS entry for the ODT-DOS EMS device driver.
- **Using Multiport Serial Cards** provides hints for using these cards with DOS processes.
- **ODT-DOS Compatibility** compares the ODT-DOS environment with a conventional, stand-alone DOS environment and discusses the requirements that DOS applications must observe in order to operate correctly with ODT-DOS.
- **Passing UNIX Environment Variables to DOS** shows you how to customize several DOS features by using UNIX environment variables.
- **Running DOS or a DOS Application as the Default Shell** tells you how to start DOS or a specific DOS application as your default environment when you log into your ODT-DOS computer.
- **Scheduling DOS Programs** illustrates how you can use the UNIX **at**, **batch**, and **nohup** commands to control the execution of DOS processes.

Running DOS In An X Window

ODT-DOS is compatible with some X Window System servers. When ODT-DOS and a compatible X server are both installed on your 386 computer, you can run UNIX and DOS commands and applications in X Window System windows. For further information on running DOS in an X Window System environment, refer to the documentation for SCO's Xsight Window System, especially the **dos** manual page.

Using Expanded Memory (EMS)

Using ODT-DOS describes expanded memory from the ODT-DOS user's perspective. Advanced users or the system administrator may need to understand the EMS device configuration command in the root-directory CONFIG.SYS file. Because this command is in the system default CONFIG.SYS file, it is interpreted whenever any user starts a DOS environment or a DOS application, unless the user specifically requests that CONFIG.SYS not be interpreted. The CONFIG.SYS command is:

```
device=c:\usr\lib\merge\emm.sys D000 208
```

In this line, "208" is the base port number used for expanded memory, and "D000" is the segment address in DOS space to which expanded memory is mapped. Do not use a value other than 208 for the base port number. However, you can use a segment address other than D000 if you have other devices that use that address. If you can't use D000 because of a conflict like this, change the EMS device configuration command in CONFIG.SYS to use one of the following segment addresses instead:

C000	C400	C800	CC00
D400	D800	DC00	E000

Using Multiport Serial Cards

Many multiport serial cards allow you to use serial ports other than COM1 and COM2 (equivalent to UNIX devices **/dev/tty1a** and **/dev/tty2a**) with DOS processes. These cards include special ODT-DOS support for the ports other than COM1 and COM2.

To use a serial port other than COM1 or COM2 with ODT-DOS, you use the **dos +a** option to attach COM1 or COM2 to your DOS process and map COM1 or COM2 to the actual UNIX name of the serial port. For example, to run a DOS environment and attach **/dev/tty20** as COM1, use the command:

```
$ dos +acom1=/dev/tty20
```

You must use indirect attachment, as illustrated in this example, to attach a serial port other than COM1 or COM2. Attempts to use direct attachment, as shown in the following example, do not work:

```
$ dos +adcom2=/dev/tty15          (incorrect)
```

Refer to *Using ODT-DOS* and *Administering ODT-DOS* for further information on using COM ports.

ODT-DOS/DOS Compatibility

The following paragraphs summarize the significant technical differences between DOS as it runs on a conventional, stand-alone personal computer and DOS as it runs under ODT-DOS. Despite these differences, nearly all DOS applications that run correctly under DOS also run correctly under ODT-DOS.

DOS Interrupts

DOS commands and applications that use standard DOS interrupt 21 function calls work properly with ODT-DOS. Commands and applications that attempt to use interrupts 13, 25, or 26 to access the shared DOS/UNIX file system do *not* work.

Commands that use interrupts 13, 25, or 26 include CHKDSK, FDISK, FORMAT, and SYS. Chapters 1 and 2 describe these commands as "exceptional" because they do not work on the shared DOS/UNIX file system.

A few applications use interrupts 13, 25, or 26 to perform low-level disk I/O operations such as:

- "Unerasing" deleted files.
- Repacking segments on the fixed disk to reduce fragmentation.
- Changing segment interleaving on the fixed disk.

You cannot use these applications to modify the shared DOS/UNIX file system.

You *can* use the commands and applications described above on a real DOS file system, such as a diskette drive, a physical or virtual DOS partition, or a virtual floppy.

If you try to use any of these commands or applications to modify the shared DOS/UNIX file system, DOS displays an error message and the operation fails, but your data and programs are not harmed in any way.

DOS Clock Interrupt Rate

By default, ODT-DOS uses the standard DOS clock interrupt rate of approximately 18 per second. All DOS applications that work at this clock interrupt rate should work in the ODT-DOS environment. ODT-DOS has an option that can change the clock interrupt rate to one per second, which improves the performance of many applications. See the description of the **merge** command **set fastclk** option in Chapter 1. The slower clock interrupt rate is compatible with most, but not all, DOS applications.

Applications That Continuously Poll The Keyboard

Unlike standard DOS, ODT-DOS automatically puts most DOS applications that continuously poll the keyboard to sleep until there is keyboard input. ODT-DOS has an option that can prevent this default behavior. See the description of the **merge** command **set pollsleep** option in Chapter 1.

Local And Remote Drives And File Handles

Unlike standard DOS, ODT-DOS treats drives that access the shared DOS/UNIX file system (C:, D:, and J:) and any DOS files contained in them as remote drives and remote files in a network. ODT-DOS has options that cause DOS to interpret these drives and files as local. See the description of the **merge** command **set drive** and **set handle** options in Chapter 1.

Limits On Number Of Open Files

DOS applications may not open more than 123 files in the shared DOS/UNIX file system using the new-style open calls (that is, require the presence in the CONFIG.SYS file of a FILES=*n* command, where *n* is greater than 123).

Passing UNIX Environment Variables To DOS

DOS and UNIX use similar concepts of environment variables (also called *environment strings*). An environment variable is a value (assigned by you, the operating system, or an application) that is available to all commands and applications that run in the same environment. Your DOS search path, for example, is available to any DOS command or application that refers to the PATH environment variable. If you type set in the DOS environment:

```
C> set
```

DOS displays the current values for all DOS environment variables, for example:

```
COMSPEC=C:\USR\DBIN\COMMAND.COM
PATH=C:\BIN;\USR\BIN;\USR\DBIN;\USR\LDBIN;
```

DOS assigns values to the PATH and PROMPT environment variables whenever you use the DOS PATH or PROMPT commands. In addition, DOS always assigns to the COMSPEC environment variable the path that DOS uses to reload the command processor (COMMAND.COM) when necessary.

DOS applications may assume that you explicitly assign the values of required environment variables yourself. You assign values to DOS environment variables with the DOS SET command. The Microsoft C Compiler, for example, assumes that you have assigned a value to the environment variable called INCLUDE, which tells the compiler where to look for #include files if they are not found in the current directory. To assign the value \USR\LDBIN\INCLUDE on DOS drive C: to the environment variable INCLUDE, type:

```
C> set include=c:\usr\ldbin\include
```

DOS environment variables are typically set in the AUTOEXEC.BAT file, so they are available for use by any command or application whenever you use DOS.

You can use DOS environment variables in the ODT-DOS environment exactly as you would use them on a conventional DOS computer. ODT-DOS, in addition, has special features for manipulating DOS environment variables that:

- Set the DOS search path automatically when you enter the DOS environment.
- Can override the default DOS search path automatically.
- Set other DOS environment variables so they are available to DOS programs run either in the DOS environment or from the UNIX shell.

These features improve DOS performance and flexibility.

Setting Your DOS Search Path With DOSPATH

ODT-DOS by default searches for DOS programs in the standard system directories **/usr/dbin** and **/usr/ldbin**. You or your system administrator can update your default path to include other directories whenever necessary (for example, when you install DOS applications). The same search path normally applies both when you use the UNIX shell and when you use the DOS environment.

Because ODT-DOS automatically sets the search path used for DOS programs, it is rarely necessary for you to change it. You can change your UNIX search path at any time using the procedures described in your UNIX manuals. By default, your new UNIX path is transmitted automatically to DOS when you enter the DOS environment.

You can, if necessary, reset the DOS search path in the DOS environment so it is different from your UNIX search path. You can reset your search path manually by using the DOS PATH command at the DOS prompt, or else override the default path in one of two ways:

1. Set your DOS search path with the PATH command in your home directory AUTOEXEC.BAT file.
2. Use the UNIX environment variable **DOSPATH** to communicate your preferred search path to DOS when you enter the DOS environment.

The second method (using **DOSPATH**) is preferred because:

- It is more efficient than running AUTOEXEC.BAT.
- Since AUTOEXEC.BAT is not always executed when you run DOS programs from the UNIX shell, any search path specified in AUTOEXEC.BAT may not be effective from the UNIX shell. The **DOSPATH** value, on the other hand, is always used.

To set your preferred DOS search path using the **DOSPATH** environment variable, follow these steps:

1. Working at the UNIX shell, type DOSPATH= followed by your preferred DOS search path. You can use either the UNIX slash (/) or the DOS backslash (\) as the path separator. If the path includes backslashes or multiple directories separated by semicolons, enclose the path in single quotes (') to prevent the UNIX shell from interpreting the backslashes or semicolons. For example:

```
$ DOSPATH='c:/usr/dbin;c:/usr/ldbin;c:/usr/mary/dbin;a:'
```

2. Export **DOSPATH**, as you would any UNIX environment variable, by typing:

```
$ export DOSPATH
```

You can use any valid DOS drive designator (such as a:, c:, d:, e:, or j:) in the **DOSPATH** definition. All directories in the shared DOS/UNIX file system that you include in your path definition must be specified using valid UNIX names, including correct use of upper- and lowercase letters. Use UNIX names rather than DOS mapped names for any directory names that are not legal in DOS. Illegal UNIX names are not included in the DOS search path.

You can include the **DOSPATH** definition and **export** instruction in your home-directory **.profile** file if you want your specified DOS path to take effect every time you log into the ODT-DOS system.

The system administrator can include a **DOSPATH** definition in **/etc/profile** that affects all users. Users can override a system **DOSPATH** with their own **DOSPATH** definition or append additional paths using a command in the form:

```
$ DOSPATH=$DOSPATH' ;newpath'
```

where *newpath* is the list of additional directories the user wants searched. For example, the **/etc/profile** file may contain the following **DOSPATH** statements:

```
DOSPATH='c:/usr/dbin;c:/usr/ldbin;c:/usr/ldbin/lotus'
export DOSPATH
```

If a user named Alan wants to keep the system **DOSPATH** but add his home directory and another subdirectory to the search path, he can add the following lines to his **$HOME/.profile**:

```
DOSPATH=$DOSPATH';c:/usr/alan;c:/usr/alan/dbin'
export DOSPATH
```

The resulting path will append Alan's personal directories to the system **DOSPATH.**

Setting Other DOS Environment Variables With DOSENV

Use the UNIX **DOSENV** environment variable to set any DOS environment variables except for PATH or COMSPEC from the UNIX shell. To use **DOSENV**, follow these steps:

1. Set the required DOS environment variables using standard UNIX syntax:

   ```
   $ VARIABLE1=VALUE1
   $ VARIABLE2=VALUE2
                .
                .
                .
   ```

2. Assign to the UNIX environment variable **DOSENV** the names of all the DOS environment variables that you defined in step 1. Use commas (but no spaces) to separate the variable names:

   ```
   $ DOSENV=VARIABLE1,VARIABLE2, . . .
   ```

3. Export each of the defined DOS environment variables and **DOSENV**:

   ```
   $ export VARIABLE1 VARIABLE2 ... DOSENV
   ```

Assume, for example, that you want to:

- Change your DOS prompt to show your current working drive and directory by using the DOS PROMPT environment variable.
- Assign the value C:\USR\LDBIN\INCLUDE to the environment variable INCLUDE, used by the Microsoft C Compiler.

To accomplish these operations, you would type:

```
$ PROMPT='$p$g'
$ INCLUDE='c:\usr\ldbin\include'
$ DOSENV=PROMPT,INCLUDE
$ export PROMPT INCLUDE DOSENV
```

Observe the following concerning this example:

- Quotes are used around pg in the first line to prevent the UNIX shell from interpreting the metacharacter "$". (For further information on the syntax for the PROMPT environment variable, refer to your DOS documentation on the PROMPT command.)
- In the second line, DOS-style backslashes are used instead of UNIX-style slashes because **DOSENV** passes the values of all variables to DOS literally, with no translation. The string "c:\usr\ldbin\include" is surrounded by single quotes to prevent the UNIX shell from interpreting the metacharacter "\".
- Items in the list of environment variables following DOSENV= are separated by commas, with no spaces.

You can define DOS environment variables using **DOSENV** in your home-directory **.profile** file to make them effective every time you log into ODT-DOS. The system administrator can include **DOSENV** definitions in **/etc/profile** that affect all users. Users can override a system **DOSENV** with their own **DOSENV** definition or append additional **DOSENV** variables to the system definition with a command in the form:

```
$ DOSENV=$DOSENV,VARIABLE1,VARIABLE2,...
```

where *VARIABLE1*, *VARIABLE2*, and so on, are the user's own preferred **DOSENV** definitions. For example, the following commands define the DOS prompt to show the current working drive and directory and add this definition to the existing system environment variable definitions.

```
$ PROMPT='$p$g'
$ DOSENV=$DOSENV,PROMPT
$ export DOSENV PROMPT
```

Using .profile Instead Of AUTOEXEC.BAT

To use DOS with greater efficiency under ODT-DOS, you should, whenever possible, use your home directory UNIX **.profile** file rather than your home directory DOS AUTOEXEC.BAT file. As described in the preceding paragraphs, you can set your DOS path and any DOS

environment variables effectively in the **.profile** file. You can also start the DOS environment or run DOS programs automatically from **.profile**.

If you can dispense with AUTOEXEC.BAT, you should use the **-p** option to disable the execution of AUTOEXEC.BAT when you use the DOS environment. See Chapter 7 for instructions on using the **±p** option.

For example, **/etc/profile** might contain the following lines to create a standard system path and prompt for the DOS environment:

```
DOSPATH='c:/usr/dbin;/usr/ldbin'
PROMPT='$p$g'
DOSENV=PROMPT
export DOSENV DOSPATH PROMPT
```

An individual user (Alan again) could further modify the DOS environment by adding the following lines to his home directory **.profile** file:

```
PROMPT='[$t$h$h$h$h$h$h] $p$g'
INCLUDE='c:\usr\alan\msc\include'
TMP='c:\usr\alan\msc\tmp'
LIB='c:\usr\alan\msc\lib'
DOSENV=$DOSENV,PROMPT,INCLUDE,TMP,LIB
DOSPATH=$DOSPATH';c:/usr/alan/dbin;c:/usr/alan/msc'
export PROMPT INCLUDE TMP LIB DOSENV DOSPATH
```

In this example, Alan uses his **$HOME/.profile** file to:

1. Override **/etc/profile**, adding a time field to his DOS prompt.
2. Set DOS environment variables (INCLUDE, TMP, and LIB) for his personal copy of the Microsoft C Compiler.
3. Append his **$HOME/dbin** and **$HOME/msc** (Microsoft C) directories to the system DOS path.

Running DOS Or A DOS Application As The Default Shell

When configured to support ODT-DOS, your computer runs the standard UNIX System V operating system. The Bourne shell is the default shell when you log in. When you type:

```
$ dos
```

ODT-DOS presents you with a continuous DOS environment — an alternative shell that understands DOS commands.

You can use the DOS shell or a specific DOS application immediately when you log in, without typing dos or the application name, by invoking DOS or the application in your home directory **.profile** file.

Running DOS As The Default Shell

To run DOS as the default shell, add the **dos** command to the end of your **.profile**, optionally followed by the **exit** command.

The **dos** command starts a DOS environment automatically as soon as you log in. You leave the DOS environment in the normal way — by typing quit. If the **dos** command is not followed with an **exit** command in your **.profile**, you return to the UNIX shell prompt when you type quit. You can then continue using the UNIX system or log out in the usual way. If the **exit** command follows the **dos** command in **.profile**, typing quit terminates your DOS environment *and* logs you out of the UNIX system.

Running A DOS Application As The Default Shell

If you want a specific DOS application to start automatically when you log into your ODT-DOS computer, follow procedures similar to those described in the previous section. For example, if you want to run Lotus 1-2-3 immediately upon login every time you use your computer, add the **123** command (instead of **dos**) to the end of your home directory **.profile**, optionally followed by the **exit** command. Then when you log in, Lotus 1-2-3 starts automatically. When you exit Lotus, your either return to the UNIX shell or are logged out, depending on whether you include the **exit** command in **.profile**.

Scheduling DOS Programs

You can use the UNIX commands **at**, **batch**, and **nohup** with DOS processes just as you use them for UNIX processes. The **at** command runs DOS or UNIX commands at a later time that you specify. The **batch** command works similarly but executes your specified processes when the system load allows rather than at a time you specify. The **nohup** command starts a process that is immune to hangups and quits.

You can run any of these commands from the UNIX shell using standard UNIX syntax and including any valid options in the DOS command. For example, you could tell the Microsoft C Compiler to compile and link the file BIG.C at 8:00 P.M. by typing:

```
$ at 2000
cl big.c
CTRL-D
```

You don't need to be logged in at the time the command executes, as long as the system is running. For further information on these UNIX scheduling commands, consult your UNIX documentation.

When you use these scheduling commands with DOS programs, observe the following precautions:

- You cannot use these scheduling commands with display-oriented DOS programs. (See the description of the ±**b** option in Chapter 7 for more on display- and stream-oriented programs.)
- Output is treated in the standard UNIX way for the scheduling command you use. If you do not redirect the output, **at** and **batch** mail your output to you. The **nohup** command causes output to be directed to the file **nohup.out** unless you specify otherwise.
- If the DOS program requires a key disk, the key disk must be in the diskette drive at the time the program runs.
- You cannot send input from your keyboard to programs started with these scheduling commands. If the program accepts redirected input, you can redirect keyboard input from a file at the time you issue the command. For example:

```
$ at 2000
doit < kbdinput > output
CTRL-D
```

Chapter 6

Tailoring The Operation Of DOS

When ODT-DOS is installed, DOS has the characteristics of DOS on a conventional personal computer with 640K bytes of memory. With this configuration, you can use DOS alone, use the ODT-DOS system as a standard UNIX computer, or combine DOS and UNIX functions. Depending on the way you use ODT-DOS and on the specific DOS commands and applications you use, you might prefer to configure DOS differently. Different DOS applications, for example, require different amounts of memory. If you use applications that require less than 640K bytes of memory, you can tell ODT-DOS how much memory you need.

This chapter and Chapter 7 (DOS Options) tell you how to modify DOS characteristics to suit your needs. The ODT-DOS system administrator can use the procedures described in these chapters to define systemwide default DOS characteristics. Individual ODT-DOS users can use the same procedures to define their own preferred DOS characteristics.

This chapter uses simple examples to explain the concepts and procedures you need to understand in order to modify DOS characteristics. Refer to Chapter 7 for complete descriptions of all the DOS options that you can use to customize the operation of DOS.

This chapter is organized as follows:

- **DOS Defaults:** How DOS defaults affect DOS environments and DOS applications.
- **Using DOS Options on the dos Command Line:** How to override DOS defaults temporarily by specifying options together with the **dos** command.
- **Installing DOS Options to Change Defaults:** How the system administrator can change DOS defaults.
- **Installing DOS Options to Override System Defaults:** How you can configure commands, applications, and the DOS environment to use your preferred DOS options automatically.
- **Using dosadmin to Install and Remove DOS Options:** How to use the **dosadmin** menu system to install and remove DOS options.
- **Using dosopt to Install and Remove DOS Options:** How to use the **dosopt** command to install and remove DOS options.

DOS Defaults

When running under ODT-DOS, DOS has many characteristics that you can customize to suit your needs. Each characteristic has a default value that you can use or change. Each DOS option described in Chapter 7 controls a specific DOS characteristic. Some of the characteristics you can control are:

- Amount of memory allocated for DOS.
- Which AUTOEXEC.BAT and CONFIG.SYS files are interpreted when you start DOS.
- Which hardware devices DOS is allowed to access.
- The initial working drive when you start DOS.

Refer to Chapter 7 for more information on specific DOS characteristics, including default values. Unless you override these defaults, they apply to:

- Any DOS environment that any user starts with the **dos** command.
- All DOS commands and applications run from the UNIX shell.[1]

Using DOS Options On The dos Command Line

Whenever you want to change a default DOS characteristic for a single execution of the DOS environment or a DOS application, use command-line options. The syntax for specifying options on the **dos** command line is:

dos *options* [*command* [*arguments*]]

[1] To improve efficiency, most of the standard DOS commands distributed with ODT-DOS have DOS options that override default values preinstalled. However, any commands or applications that you add yourself use default values unless you explicitly override them.

For example, to run the DOS environment with 256K bytes of memory, type:

```
$ dos +m256
```

Similarly, to run WordStar with 256K bytes of memory to create a file called **report**, type:

```
$ dos +m256 ws report
```

Options consist of a single alphabetic character (**m** in the previous examples) preceded by either a plus or minus sign (+ or -). The plus or minus sign generally turns the characteristic controlled by that option "on" or "off." The **+x** option, for example, means "turn DOS break-checking on," while **-x** means "turn DOS break-checking off."

Some options require or are allowed to take parameters. The number "256" in the command **dos +m256**, for example, is a parameter to the **+m** option that specifies the number of kilobytes of memory to allocate.

Options that do not take parameters can be grouped together following a single plus or minus:

```
$ dos +bc -tx compute
```

Options that can take parameters must be specified separately:

```
$ dos +e/usr/dave/config.sys +l/usr/dave
```


The order of options is immaterial. The following two examples are both valid and both have the same effect:

```
$ dos +x -e -s
$ dos -s +x -e
```

DOS options that you specify on the **dos** command line override DOS defaults and also override any DOS options that are installed in an application or options file.

Installing DOS Options To Change Defaults

The system administrator can redefine DOS defaults by installing DOS options in either of the two ODT-DOS options files **/etc/dosenv.def** and **/etc/dosapp.def**. The file **/etc/dosenv.def** controls the default

characteristics of DOS environments that users start with the **dos** command. The file **/etc/dosapp.def** controls the default characteristics of DOS commands and applications run from the UNIX shell.

There are two ways the system administrator can change DOS defaults:

- Use the **dosadmin** menu system, which provides convenient ways to change default values for the DOS characteristics that most often need changing.
- Use the **dosopt** command, which is less convenient but more flexible than **dosadmin**.

Both **dosadmin** and **dosopt** are described more thoroughly later in this chapter.

Installing DOS Options To Override System Defaults

You can override DOS defaults by installing DOS options in personal options files, DOS commands, and DOS applications. You should install DOS options to override system defaults whenever you want to override a default value more permanently than you can by using DOS options on the **dos** command line. Installed options remain effective until you change or override them. Command-line options are effective only for one execution of a DOS environment or application.

Users can install DOS options in their personal options files **dosenv.def** and **dosapp.def**, which must be in their home directories. These personal options files do not exist until users create them.[1] Like the analogous files in the **/etc** directory, the home directory **dosenv.def** controls the characteristics of DOS environments, and the home directory **dosapp.def** controls the characteristics of DOS commands and applications run from the UNIX shell.

The values defined in these home directory files are effective only for the user who owns the files. Values defined in home directory files override values defined in the options files in **/etc**. Home directory options files are fully specified. That is, values for every configurable DOS characteristic are defined.

[1] **dosadmin** creates personal options files automatically when required. Also see "Creating Home Directory Options Files," later in this chapter.

You can also configure specific DOS applications (Lotus 1-2-3, for example) so they use your preferred values when you run them from the UNIX shell. When you install specific DOS options directly in executable DOS command and application files, the options override DOS defaults and options installed in options files. They can be superseded only by using DOS options on the **dos** command line. You don't need to fully specify all DOS characteristics when you install options in a command or application. Unspecified characteristics take their values from the home directory **dosapp.def**, if it exists, or from **/etc/dosapp.def** if it doesn't.

Procedures for installing options in personal options files or in applications are generally the same as the procedures for changing defaults in **/etc/dosenv.def** and **/etc/dosapp.def**. You can use the **dosadmin** menu system or the **dosopt** command as described in the following sections.

Using dosadmin To Install And Remove DOS Options

The **dosadmin** menu system provides an easy way to change the values of three important DOS characteristics: memory, DOS startup files, and DOS device files. The following paragraphs tell you more about these characteristics and how to modify them using **dosadmin**.

Characteristics You Can Tailor Using dosadmin

You can use **dosadmin** to install DOS options that affect memory, DOS startup files, and DOS device files. Depending how you use **dosadmin**, you can tailor these characteristics for:

- The DOS environment.
- All DOS commands and applications run from the UNIX shell.
- Specific DOS commands and applications.

Startup Memory

The ODT-DOS factory default for DOS memory is 640K bytes. This means that, by default, the DOS environment, DOS applications run in the DOS environment, and DOS applications run from the UNIX shell all have 640K bytes of memory available to them. To specify your preferred amount of memory, you can enter a value between 128 and 640 (to specify number of kilobytes).

ODT-DOS runs more efficiently if you allocate only as much memory as you need. For example, if you normally need only 384K bytes of memory while in the DOS environment, you can install the 384K value as your preferred memory value.

DOS Startup File

On a conventional DOS computer, if you have a file called \AUTOEXEC.BAT on your system disk, the commands in that file are executed automatically whenever you boot the computer. When you use ODT-DOS, you don't boot DOS the same way you would on a stand-alone personal computer. ODT-DOS, however, can execute an AUTOEXEC.BAT file when you use the DOS environment or run a DOS program, just as though you had booted DOS on a conventional personal computer.

The ODT-DOS factory default is to execute *two* AUTOEXEC.BAT files in succession when you run the DOS environment by typing dos. These two files are the root directory AUTOEXEC.BAT and your home directory AUTOEXEC.BAT.

When you run DOS programs from the UNIX shell, the ODT-DOS factory default is not to run *any* AUTOEXEC.BAT file.

Using **dosadmin**, you can specify a single DOS batch file that is to be executed instead of the default files when you start the DOS environment or a DOS program, or specify that *no* batch file is to be executed. The file you specify can have any legal DOS or UNIX file name, and you can specify it either by full UNIX path name, for example:

```
/usr/joe/dbin/appl.bat
```

or by DOS drive and full DOS path name, for example:

```
c:\usr\joe\dbin\appl.bat
```

DOS Device File

On a conventional DOS computer, the file \CONFIG.SYS is interpreted automatically when you boot DOS. ODT-DOS by default treats CONFIG.SYS files similarly to AUTOEXEC.BAT files: it interprets the root directory CONFIG.SYS file and your home directory CONFIG.SYS file in succession whenever you run the DOS environment or a DOS application.

Using **dosadmin**, you can specify *one or more* configuration files that are to be interpreted instead of the defaults, or you can specify that *no* configuration file is to be interpreted. The files you specify can have any legal UNIX file name. You can specify files either by full UNIX path, for example:

```
/usr/joe/dbin/config.sys
```

or by DOS drive and full DOS path name, for example:

```
c:\usr\joe\dbin\config.sys
```

To specify multiple files, list them in the order they are to be interpreted, separated by a space, a comma, or both. For example:

```
c:\config.sys, d:\special.sys
```

Using dosadmin To Tailor DOS

NOTE: When you use **dosadmin** to tailor DOS, ODT-DOS maintains a record of your configuration in the file **dosenv.def** or **dosapp.def**. These files are stored in your home directory if you are a system user or in /etc if you are the system administrator logged in as **root**. You don't need to be concerned with the contents of these files, but be sure you don't accidentally delete them.

To use **dosadmin** to tailor DOS, follow these procedures:

- Select the appropriate Configuration menu.
- Identify the DOS application you are tailoring, if you are tailoring a specific application.
- Fill in the fields for memory, startup file, and device file.

The following paragraphs describe these steps in more detail.

Selecting The Configuration Menu

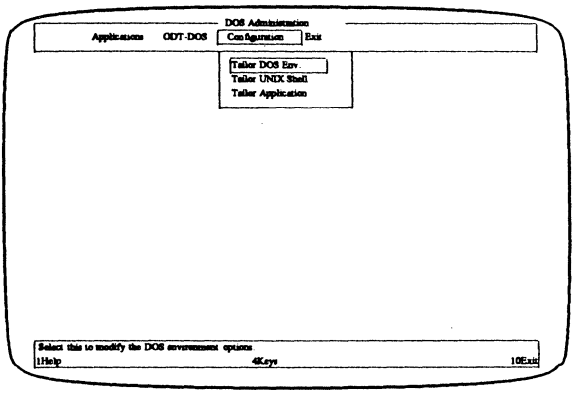
Follow these steps to select the appropriate Configuration menu:

1. Type:[1]

```
$ dosadmin
```

   The main DOS Administration menu is displayed.

2. Press the left or right arrow key (← or →) to move the highlighted field at the top of the screen to Configuration. Your display looks like this:



   The Configuration menu offers three selections:

   - Tailor DOS Env.
   - Tailor UNIX Shell
   - Tailor Application

   Use the up or down arrow key (↑ or ↓) to highlight the appropriate field and press ENTER (↵). Follow these guidelines to make your selection:

   - **Tailor DOS Env.**: Use this selection when you want to modify the characteristics of the DOS environment (that is, the DOS characteristics that apply when you type dos). If you are logged in as **root**, the characteristics you specify *replace* existing DOS

[1] You can type the **dosadmin** command either at the UNIX shell prompt or in the DOS environment.

defaults for all users who run DOS environments. If you are not logged in as **root**, the characteristics you specify *override* existing DOS defaults when *you* run a DOS environment.

You can override characteristics specified with this menu by using DOS command-line options when you use the **dos** command as described earlier in this chapter. Characteristics that you specify using this menu do not affect DOS commands or applications run from the UNIX shell.

- **Tailor UNIX Shell**: Use this selection when you want to modify the characteristics of DOS commands and applications when they are run from the UNIX shell. If you are logged in as **root**, the characteristics you specify *replace* existing DOS defaults for all users who run DOS commands and applications from the UNIX shell. If you are not logged in as **root**, the characteristics you specify *override* existing DOS defaults when *you* run DOS commands or applications from the UNIX shell.

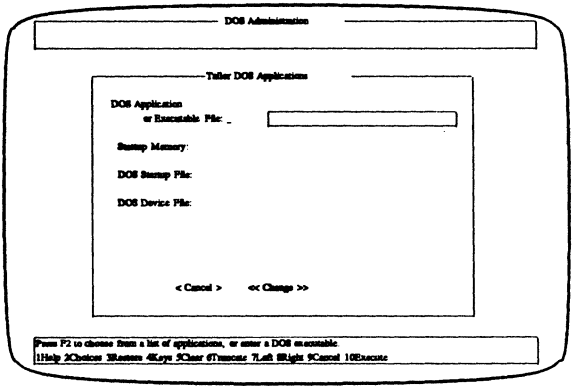
  Any characteristics you specify using this menu can be overridden for specific applications by using the Tailor Application menu or DOS command-line options.

- **Tailor Application**: Use this selection when you want to modify the characteristics of a specific DOS command or application when it is run from the UNIX shell. If you are logged in as **root**, you can tailor any command or application, although normally you use this menu as **root** in order to tailor publicly accessible applications in directories such as **/usr/ldbin**. If you are not logged in as **root**, you can tailor only commands and applications you have write permission for.

If you have selected the Tailor Application menu, you must identify the command or application you want to tailor as described in the next section. If you have selected the Tailor DOS Environment or Tailor UNIX Shell menu, skip ahead to "Filling in the Configuration Menu."

Selecting The DOS Command Or Application

If you have selected the Tailor Application menu, your display resembles this:



You must now fill in the "DOS Application or Executable File" field to identify the command or application you want to tailor. You can fill in this field in three different ways:

- Type in the name of the application exactly as you entered it when you installed the application using **dosadmin** (for example, "Lotus 1-2-3 (lotus)"). (See *Administering ODT-DOS*.)

- Type in the name of a DOS executable file. You can name the file by full UNIX path or by DOS drive and full DOS path, for example:

  /usr/ldbin/lotus.com

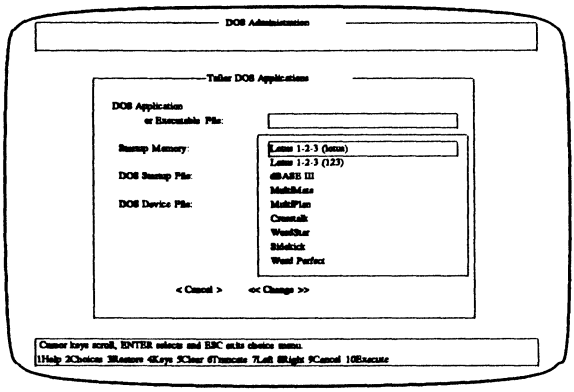
  or

  c:\usr\ldbin\lotus.com

  You should use this method to specify a DOS executable file if you are tailoring a program that you have not installed using **dosadmin** and is not in the **dosadmin** database.

- Press the F2 key to display a list of applications currently installed in the **dosadmin** database, and select from this list by pressing ENTER.

The last method listed above is convenient because you don't have to remember the exact application name as you entered it in the **dosadmin** database. If you press F2 while the "DOS Application or Executable File" field is highlighted, your display shows all applications in the **dosadmin** database, like this:



Use the up or down arrow keys (↑ or ↓) to move the highlighted field up or down in this list until you get to the application you want to tailor. If there are more applications than there is room to display them, you can press the up or down arrow key to scroll the list and expose more application names.

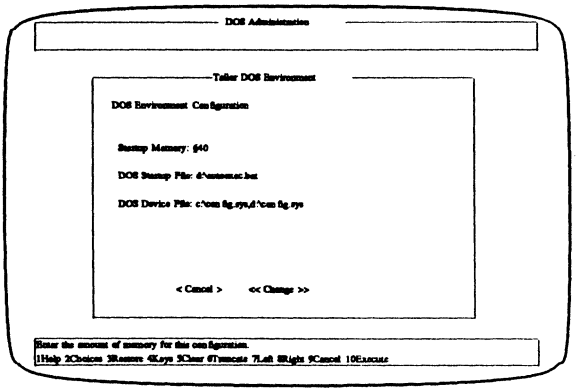
When the application you want to tailor is highlighted, press ENTER. The list of applications then disappears and your chosen application name is automatically entered in the "DOS Application or Executable File" field.

Continue with the steps described in the next section.

6

Filling In The Configuration Menu

The Tailor DOS Environment, Tailor UNIX Shell, and Tailor Application menus are similar. They all have fields named "Startup Memory," "DOS Startup File," and "DOS Device File." The display resembles this:



The values shown for "Startup Memory," "DOS Startup File," and "DOS Device File" are the current values that apply to the DOS environment, application, or applications you are configuring. The actual values shown on your display may differ from those in this illustration.

To change any of the displayed values, press the TAB key (⇆) until the field you want to change is highlighted, and then type in the value you want. If you make a mistake, press the BACKSPACE key (←) and retype your entry or press the F5 key to clear the field and then retype your entry.

When the "DOS Startup File" or "DOS Device File" field is highlighted, you can use the F5 key to clear the field, which results in no startup file or device file being interpreted when you start the DOS environment.

Follow these guidelines as you fill in each field:

- **Startup Memory**: Use any value between 128 and 640 (in kilobytes).
- **DOS Startup File**: If this field is filled in *only* with d:\autoexec.bat, it means *both* the **root** directory AUTOEXEC.BAT and personal, home-directory AUTOEXEC.BAT files will be executed.[1] If you want only one AUTOEXEC.BAT file to be interpreted, identify it using DOS drive C: or a UNIX-style path name. For example, filling in this field with c:\usr\greg\autoexec.bat causes only Greg's home-directory AUTOEXEC.BAT to be executed. Filling in /autoexec.bat causes only the root-directory AUTOEXEC.BAT to be executed.

  You are not restricted to files named AUTOEXEC.BAT for this field. You can identify any single DOS file, in any directory in the shared DOS/UNIX file system, that contains legal DOS batch commands.
- **DOS Device File**: This field identifies the device configuration files that you want DOS to interpret. It normally shows both c:\config.sys and d:\config.sys. (Refer to Chapter 1 for more information on drive D: and other DOS drives.) When these two file names are displayed, DOS interprets both the root directory CONFIG.SYS and a home-directory CONFIG.SYS.[2]

  You can change or delete the files displayed in this field. For most purposes, however, it is preferable to let DOS interpret the default CONFIG.SYS files.

  You are not restricted to files named CONFIG.SYS for this field. You can identify one or more DOS files, in any directories in the shared DOS/UNIX file system, that contain legal DOS CONFIG.SYS commands.

When you have filled in the configuration menu, check your entries for accuracy. If you choose to cancel the tailoring operation, press the TAB key until the "Cancel" field is highlighted and press ENTER.

When you have entered the values you want, press the TAB key until the "Change" field is highlighted and press ENTER to save the new values you have specified.

[1] If you are logged in as root when you use DOS, however, only one AUTOEXEC.BAT file is executed — the one in the root directory.

[2] If you are logged in as root when you use DOS, however, only one CONFIG.SYS file is executed — the one in the root directory.

The top-level DOS Administration menu reappears. Press the left or right arrow key until "Exit" is highlighted, and press ENTER. Your UNIX or DOS system prompt returns.

Using dosopt To Install And Remove DOS Options

You use the **dosopt** command (like you use the **dosadmin** menu system) to assign DOS characteristics such as memory, startup file, and device configuration file to applications or to your environment. The **dosopt** command is more flexible than **dosadmin**. You can use it to assign many characteristics besides memory, startup file, and configuration file. You can also use **dosopt** where you can't use **dosadmin** — inside UNIX shell scripts, for example.

As with **dosadmin**, you can use the **dosopt** command to install DOS options that determine:

- Default characteristics of DOS environments for all users.
- Characteristics of DOS environments for individual users who want to override the default characteristics.
- Default characteristics of applications run from the UNIX shell by all users.
- Characteristics of applications run from the UNIX shell by individual users who want to override default characteristics.
- Characteristics of specific DOS applications run from the UNIX shell.

The syntax for **dosopt** is:

dosopt *options options_file*

or

dosopt *options application*

As described earlier, under "Using DOS Options on the **dos** Command Line," *options* are key letters preceded by a plus or minus (+ or -) sign that identify the values of specific DOS characteristics. Chapter 7 describes each DOS option thoroughly.

options files are files that hold records of DOS options you have installed. Different options files define system defaults and user preferences for the DOS environment and for DOS applications. In the command form **dosopt** *options application*, *application* is the name of a specific DOS application — "123," for example — in which you want to install DOS options.

The names of the options files and the DOS processes they affect are:

- **/etc/dosenv.def**: This file defines system default characteristics for DOS environments started by all users. DOS options defined in this file can be overridden by using DOS options on the **dos** command line or by installing DOS options in home directory **dosenv.def** files.

- **$HOME/dosenv.def**: A user's home directory **dosenv.def** file overrides *all* values defined in **/etc/dosenv.def** and defines characteristics of DOS environments started by that user. DOS options defined in **$HOME/dosenv.def** can be overridden by using **dos** command-line options.

- **/etc/dosapp.def**: This file defines system default characteristics for DOS applications invoked from the UNIX shell. DOS options defined in this file can be overridden by using **dos** command-line options, by installing options in a home directory **dosapp.def** file, or by installing options in specific applications.

- **$HOME/dosapp.def**: A user's home directory **dosapp.def** file overrides *all* values defined in **/etc/dosapp.def** and defines characteristics of applications started from the UNIX shell by that user. DOS options defined in **$HOME/dosapp.def** can be overridden by using **dos** command-line options or by installing options in specific applications.

- **A specific DOS application**: Although not an options file in the same sense as the files listed above, any DOS executable file can hold a record of DOS options that apply to that file when you run it from the UNIX shell. Options installed in DOS commands and applications override **/etc/dosapp.def** and **$HOME/dosapp.def**. You can override options installed in commands and applications by using **dos** command-line options.

Only the system administrator, logged in as **root**, can change options in **/etc/dosenv.def** and **/etc/dosapp.def**. All users can create **dosenv.def** and **dosapp.def** files in their home directories and install options in them. Anyone who has write permission for a command or application can install options in the executable file.

Creating Home Directory Options Files

The system default options files **/etc/dosenv.def** and **/etc/dosapp.def** are created automatically when you install ODT-DOS. Home directory **dosenv.def** and **dosapp.def** files, however, do not exist until you create them. There are two ways to create these home directory files:

- Use the **dosadmin** menu system as described earlier in this chapter to modify the characteristics of the DOS environment or DOS applications. If you are logged in as a user who has a home directory, **dosadmin** automatically creates **$HOME/dosenv.def** the first time you use the "Tailor DOS Env." menu to tailor the DOS environment. **dosadmin** automatically creates **$HOME/dosapp.def** the first time you use the "Tailor UNIX Shell" menu to tailor the operation of commands and applications from the UNIX shell.

- Copy **/etc/dosenv.def** and **/etc/dosapp.def** to your home directory and make sure they are writable. If you are working in your home directory, you can use the following commands:

```
$ cp /etc/dosenv.def dosenv.def
$ cp /etc/dosapp.def dosapp.def
$ chmod u+w dosenv.def dosapp.def
```

Installing DOS Options

Following are a few examples of **dosopt** commands to illustrate how you install options in options files and in applications. For more information on the meaning of specific options, refer to Chapter 7.

If you are working in your home directory, the command:

```
$ dosopt +m512 +x dosenv.def
```

overrides the default values for memory (**+m**) and DOS break-checking (**+x**) so that you get 512K bytes of memory and DOS break-checking is "on" whenever *you* run a DOS environment. If the system administrator enters the following similar command:

```
# dosopt +m512 +x /etc/dosenv.def
```

these values become the defaults for all users who run DOS environments.

Preceding a DOS option with a minus sign (-) generally turns off that function. For example, to prevent AUTOEXEC.BAT from running, type:

```
$ dosopt -p dosenv.def
```

You can change values at different times without affecting values you specified previously:

```
$ dosopt -p dosenv.def
$ dosopt +x dosenv.def
```

In this example, AUTOEXEC.BAT is disabled and DOS break-checking is turned on in two separate commands. The AUTOEXEC.BAT file remains disabled unless it is later explicitly enabled with the **+p** option. If you are working in your home directory, the command:

```
$ dosopt -s dosapp.def
```

overrides the default for the DOS printer timeout and disables the timeout when *you* run DOS applications from the UNIX shell.

The command:

```
$ dosopt +b /usr/vicky/dbin/cl
```

identifies the **cl** program as stream-oriented.

Note that DOS applications that you can run from the UNIX shell normally have two names: a DOS file name with an extension (**.exe**, **.com**, or **.bat**) and a UNIX name without an extension, which is linked to the DOS name. For example, the file **cl.exe** would be linked to **cl**. When you use **dosopt** to install an option in an application, you can use either name. That is, the following two commands have the same effect since **cl.exe** and **cl** are two names for the same file:

```
$ dosopt +b cl
$ dosopt +b cl.exe
```

Displaying Installed Options

You can display the options currently installed in your home-directory **dosenv.def** or **dosapp.def** files by typing:

```
$ dosopt +v dosenv.def
$ dosopt +v dosapp.def
```

(You must be working in your home directory, or else supply the path names of the **dosenv.def** and **dosapp.def** files.) The command:

```
$ dosopt +v
```

is an abbreviation for **dosopt +v $HOME/dosenv.def**.

To display the current system default DOS options, issue one of these commands:

```
$ dosopt +v /etc/dosenv.def
$ dosopt +v /etc/dosapp.def
```

Removing Installed Options

To remove an option installed in a command or application, use the **dosopt +z** option. For example, if you have assigned 512K bytes of memory to Lotus 1-2-3 using the command:

```
$ dosopt +m512 ws
```

you can use the **+z** option to remove the installed memory value and allow the memory value from **$HOME/dosapp.def** (if it exists) or **/etc/dosapp.def** to be effective. The command is:

```
$ dosopt +zm ws
```

When you use **+z** to remove previously installed **+a** ("attach device") options, you can remove individual **+a***device* options or you can remove *all* **+a** options with a single **dosopt** command. For example, assume you have used the following **dosopt** commands to specify attachment of both COM1 and COM2 for the application named APPL:

```
$ dosopt +acom1 appl
$ dosopt +acom2 appl
```

To remove the **+acom2** instruction but keep the **+acom1** instruction, type:

```
$ dosopt +zacom2 appl
```

To remove all **+a** options from the file APPL at once, type:

```
$ dosopt +za appl
```

Refer to Chapter 7 for further information on the **±a** option.

To remove *all* installed options in a DOS command or application, use an uppercase "Z":

```
$ dosopt +Z appl
```

This command removes all installed options from **appl** and allows the option values in **$HOME/dosapp.def** (if it exists) or **/etc/dosapp.def** to be effective.

The **+z** and **+Z** options cannot be used on the options files **$HOME/dosenv.def**, **$HOME/dosapp.def**, **/etc/dosenv.def**, or **/etc/dosapp.def**.

Restoring DOS Options Files

If for any reason you want to restore the original contents in either of your system default DOS options files (**/etc/dosenv.def** or **/etc/dosapp.def**), you can copy the corresponding file from the directory **/usr/lib/merge**. **/usr/lib/merge/dosenv.def** and **/usr/lib/merge/dosapp.def** are permanent copies of **/etc/dosenv.def** and **/etc/dosapp.def** in the form that exists when you install ODT-DOS.

Chapter 7
DOS Options

This chapter explains the options you can use to modify the operation of DOS. To use these options effectively, you should be familiar with the procedures for using the **dosopt** and **dos** commands described in Chapter 6. You can use the options described in this chapter with the **dosopt** command to install options in options files (**/etc/dosenv.def**, **$HOME/dosenv.def**, **/etc/dosapp.def**, and **$HOME/dosapp.def**) and in DOS applications. You can use the same options on the **dos** command line when you don't want to install permanent values.

The following table lists the options and summarizes their use.

Following the table are complete descriptions of each option, arranged alphabetically by option name. Each description includes a "Usage" note, showing the possible ways of using that option with **dos** or **dosopt**. Type all boldface letters and words exactly as shown. The symbol ± means you can use the option with either a + or a - (usually to turn the indicated value on or off). Italicized words (like *file*) are generic terms for which you should substitute actual names (such as the actual name of a file on your computer). Brackets surrounding data (such as [*file*]) mean that supplying the bracketed data is optional. A vertical bar (|) means "either or." Choose one of the separated items and type it as part of the command. For example, **dosopt +v**[*options_file* | *program*] means you may name either an options file or a DOS program when you use the +v option.

Unless otherwise indicated, you can use any option in the following ways:

- With the **dosopt** command to install your preferred values in any of the following options files:
  - **/etc/dosenv.def**
  - **$HOME/dosenv.def**
  - **/etc/dosapp.def**
  - **$HOME/dosapp.def**
- With the **dosopt** command to install your preferred values in a DOS application.
- On the **dos** command line.

For example, the usage line:

+d[*drive*]

means any of the following commands are allowed:

```
# dosopt +dd /etc/dosenv.def
# dosopt +dd /etc/dosapp.def
$ dosopt +dd $HOME/dosenv.def
$ dosopt +dd $HOME/dosenv.def
$ dosopt +dd cl
$ dos +dd
$ dos +dd cl temp.c
```

You can substitute any legal DOS drive letter for the second "d" in +dd in each of these examples. You can also omit the drive letter to indicate the default (drive C:).

Each option description also includes a "Factory default" note that shows the original system default value (defined in **/etc/dosenv.def** and **/etc/dosapp.def**) when ODT-DOS is installed.

Option	Meaning
+a*dos_device*	Attach specified DOS device.
-a*dos_device*	Cancel DOS device attachment.
+b	DOS program is stream-oriented.
-b	DOS program is display-oriented.
+c	Pass command unchanged to **command.com**[1]
+d[*drive*]	Set initial current drive.
+e[*file*[,*file* ...]]	Interpret one or more *files* instead of default configuration files. If *files* are not specified, default **/config.sys** and **$HOME/config.sys** are used.
-e	Don't interpret any **config.sys** when DOS is run.
+h	Display help text.
+l[*file*|*directory*]	Load DOS image *file* or an image file from *directory* instead of default image.
-l	Don't use a DOS image. Boot from drive A: instead.
+m*n*	Allocate *n*K bytes of memory.
+p[*file*]	Run **/autoexec.bat** and **$HOME/autoexec.bat**. *file*, if specified, is the alternative batch file to be run instead of the defaults.
-p	Do not run any **autoexec.bat**.
+s[*n*]	Print accumulated DOS printer output when no characters have been sent to the printer for *n* seconds. *n* is a timeout value between 5 and 3600 seconds (default 15 seconds).
-s	Don't print DOS printer output until DOS process exits.
+t	Translate DOS switch characters and path separators in standard way.
-t	Don't translate DOS switch characters and path separators.
+v	"Verbose" mode: display **dosopt** acknowledgement message.[2]
+x	DOS break-checking on.
-x	DOS break-checking off.
+y	Null option for DOS .COM files that have no other options.[2]
+z	Remove an option from an application.[2]
+Z	Remove all options from an application.[2]

[1] The +c command can be used only on the **dos** command line. It cannot be used with **dosopt**.

[2] The +v, +y, +z, and +Z options can be used only with **dosopt**. They cannot be used on the **dos** command line.

Attach Devices To DOS

Usage: ±a*device*
-a[*device*]

Factory default: Attach default devices automatically

There are several ways you can configure the hardware devices that DOS processes can access. ODT-DOS automatically manages access to the most important hardware devices, such as the keyboard, the display, and diskette drives. You can use the ±a option to alter the default ODT-DOS treatment of some of these devices. There are also several devices that are not available automatically when you use DOS because they are shared with UNIX and other DOS processes. You must explicitly request access to these devices using the +a option.

Default Devices

ODT-DOS automatically makes the following devices available to DOS processes:

- Console and terminal keyboards
- The fixed disk
- Diskette drives A: and B:
- A mouse
- The physical DOS partition
- Console and terminal displays

The Keyboard

Communication between DOS processes and console and terminal keyboards is transparent and automatic.

The Fixed Disk

The shared DOS/UNIX file system on the fixed disk, including DOS drives C:, D:, and J:, is also available automatically to all DOS processes.

Diskette Drives

Diskette drives A: and B: are available to DOS as long as they are not mounted as UNIX devices or otherwise used by UNIX (by a **cpio** process, for example). The diskette drives are assigned on a first-come-first-served basis to any DOS process that attempts to access them whenever they have been unused for five seconds or longer. Drives A: and B: are assigned and released simultaneously.

The Mouse

Any mouse that is properly installed and accessible to UNIX is also accessible to DOS (provided it is not actively being used by another process at the moment DOS tries to use it). DOS processes always view the mouse as a Microsoft bus mouse. There is a device driver entry in the default ODT-DOS **/config.sys** file to support DOS communication with a mouse. When you use the MultiScreen™ feature on the console, the mouse automatically communicates with the screen currently displayed.

The Physical DOS Partition

The physical DOS partition is automatically available as drive E: to all DOS processes. Although any number of DOS processes can read drive E: simultaneously, only one process can write to it at one time. When any process writes to drive E:, it is unavailable to other processes for both reading and writing until the process that started writing exits. Unlike the shared DOS/UNIX file system, the physical DOS partition is optional and may not exist on your computer. If a DOS partition exists on your fixed disk when you install ODT-DOS, the installation procedure configures it for ODT-DOS use as the device named **/dev/dsk/dos**. The files **/etc/dosenv.def** and **/etc/dosapp.def** both contain the following entry to make **/dev/dsk/dos** accessible as drive E:

```
+ae:=/dev/dsk/dos
```

These entries can be changed or overridden. For more information on the syntax used in this command, see "Attaching DOS Drives," later.

Display Adapters

ODT-DOS supports standard ASCII terminals, PC scancode terminals, and consoles using standard monochrome (MDA), CGA, EGA, VGA, or Hercules display adapters. In normal use, ODT-DOS automatically sets up your DOS environment to accommodate the type of console or terminal you are using when you run DOS.

You can explicitly specify monochrome, CGA, EGA, VGA, or Hercules capabilities using the **+a** option, although there is typically no need to do so. Your display must have at least the capabilities reflected by the option you select — for example, you cannot specify a CGA adapter type if you are using an ASCII terminal or a console that has an MDA adapter. The following table summarizes the display adapters you can specify on each type of terminal and the resulting capabilities:

	Capability By Physical Display Type				
Option	**Monochrome or Serial Terminal**	**Hercules**	**CGA**	**EGA**	**VGA**
None	mono	herc	CGA	EGA	VGA
+amono	mono	mono	mono	mono	mono
+aherc	-	herc	-	-	-
+acga	-	-	CGA	CGA	CGA
+aega	-	-	-	EGA	-
+arega	-	-	-	EGA	-
+avga	-	-	-	-	VGA

ODT-DOS has two different EGA modes. The default (**+aega**) runs faster but may corrupt the displayed image when you use the MultiScreen feature and switch screens while graphic images are being calculated or painted. The *reliable* EGA option, **+arega**, causes DOS to run slower on an EGA display but eliminates the risk of screen corruption.

If you use the MultiScreen feature with either an EGA or a VGA display and switch away from a screen displaying a high-resolution graphics application, the application is suspended. Applications running in monochrome, CGA, or Hercules mode continue to run even when their screens are not visible.

Other Standard DOS Devices

The devices described in this section must be explicitly requested using the **+a** option. The specific **+a** syntax that applies to each device is shown together with the device description.

Except for expanded memory (EMS), these devices are available to only one process at a time. If DOS attempts to access a device that is currently in use by another DOS or UNIX process, ODT-DOS displays an error message and aborts the DOS process.

COM Ports

Usage: **+acom1**[=*unix_device*]
+acom2[=*unix_device*]
+adcom1[=**dcom2**]
+adcom2[=**dcom1**]

You can attach COM ports to DOS processes either directly (**+adcom1** or **+adcom2**) or indirectly (**+acom1** or **+acom2**). When you use direct attachment, DOS controls the COM hardware directly. When you use indirect attachment, ODT-DOS mediates the communication between DOS and the COM hardware. Refer to *Using ODT-DOS* for further information on the advantages and disadvantages of both forms of attachment.

unix_device is the name of any terminal line in the form **/dev/tty***xx*, where **tty***xx* is a specific name such as **tty1a** or **tty20**. When you attach a COM port to a DOS process, the specified terminal line is accessible to DOS as COM1 or COM2. By default, **+acom1** means **+acom1=/dev/tty1a** and **+acom2** means **+acom2=/dev/tty2a**. If your computer has a compatible multiport serial card, you can attach other terminal lines as COM1 or COM2 by using options such as **+acom2=/dev/tty20**.

Use the option form **+adcom***n*=**dcom***n* when you want a DOS application to view one COM port as though it is a different port. For example, the option **+adcom1=dcom2** causes DOS to treat the physical COM2 port as though it is COM1.

Expanded Memory (EMS)

Usage: **+aems**[*n*]

ODT-DOS simulates DOS expanded memory as specified by Lotus, Intel, and Microsoft.[1] **+aems** requests the default of one megabyte of expanded memory. To request a different amount, use **+aems***n*, where *n* is 512, for 512K bytes of expanded memory, or 1, 2, 3, 4, 5, 6, or 8, specifying megabytes.

There is a device driver entry for EMS in the system default **/config.sys** file, so EMS is available only if DOS interprets this file or another **config.sys** file that has a similar entry.

Each DOS process can request its own private expanded memory, so multiple DOS processes and multiple users can use EMS simultaneously.

Refer to Chapter 5 and to *Using ODT-DOS* for more information on expanded memory.

Printer Ports

Usage: **+alp***n*

By default, ODT-DOS sends all DOS printer output to the UNIX print spooler, so DOS does not require access to printer ports. If you want DOS to control a printer port directly, attach the port using the **+alp***n* option, where *n* is 0, 1, or 2. Refer to Chapter 3, Advanced Printing, for further information.

The Game Port

Usage: **+agame**

To use the game port with DOS, use the **+agame** option.

Attaching DOS Drives

Usage: **+a***drive_letter*:=*unix_file_name*[**e,i,r**]

As mentioned earlier, the shared DOS/UNIX file system, including DOS drives C:, D:, and J:, is always available to DOS, and ODT-DOS automatically manages access to diskette drives.

---

[1] *The Lotus/Intel/Microsoft Expanded Memory Specification*, Version 3.20, Part number 300275-003, September 1985, Lotus Development Corporation, Intel Corporation, and Microsoft Corporation.

A physical DOS partition, if your computer has one, is accessible as drive E:. The **+a** option **+ae:=/dev/dsk/dos**, which is included in the default **/etc/dosenv.def** and **/etc/dosapp.def** files, is required to make the DOS partition accessible as drive E:.

The **+ae:=/dev/dsk/dos** option is one example of a general option form that you can use to access the physical DOS partition, virtual DOS partitions, and virtual DOS floppies. *Administering ODT-DOS* tells you how to create virtual partitions and floppies. To access any of these devices with DOS, you use the **+a** option and specify a DOS drive and a UNIX file name. The UNIX file name identifies the file that contains the DOS partition or virtual floppy. The DOS drive letter can be any legal DOS drive name. Follow these guidelines when you select a DOS drive letter:

- Use drive name a or b to access a virtual floppy.
- If you attach a virtual floppy using either a or b as the drive letter, you cannot simultaneously access a physical diskette drive with the same name. You can, however, access physical and virtual floppies with different names at the same time.
- Use a drive name higher than b to access the physical DOS partition or a virtual DOS partition.
- Drives C:, D:, and J: have specific functions in the ODT-DOS environment, so avoid using these names for virtual partitions.
- The physical DOS partition (**/dev/dsk/dos**) is accessible as drive E: by default. You can use the drive name e to access a virtual partition, but if you do, the physical partition is not simultaneously accessible as drive E:. You can also use **+a** with a drive letter other than e to access **/dev/dsk/dos** if you choose.
- The default ODT-DOS LASTDRIVE is "n", so if you want to use a drive name later in the alphabet, redefine LASTDRIVE in a **config.sys** file that is interpreted when you start DOS.

Virtual partitions and floppies have the same limitations as the physical DOS partition. Multiple DOS processes can read them at the same time, but only one process can write to each of these devices at a time. When one process starts writing, other processes are prevented from both reading and writing to that device until the process that started writing exits.

Specifying Immediate, Exclusive, Or Read-Only Access

When you request access to a DOS drive using the +a*drive_letter*:=*unix_file_name* option, ODT-DOS does *not* immediately attach the specified device to your DOS process. Instead, ODT-DOS determines whether the device is available, and attaches it if it is, only when DOS attempts to read or write to the device. If the requested device is unavailable when DOS attempts to access it, the DOS process may display an error message or behave incorrectly. To avoid these problems, you can request that the device be attached to your DOS process immediately or exclusively.

To specify immediate attachment, include the **i** key letter following the UNIX file name. The DOS process then aborts immediately if the specified device is unavailable (for example, if another process is writing to it or it doesn't exist).

The **e** key letter requests exclusive access to the specified device so that no other process can access it. This option is useful whenever you want to prevent other processes from reading or writing to a device while you use it. The **e** key letter automatically implies the **i** key letter. That is, exclusive attachment also means immediate attachment. It is therefore unnecessary to specify **i** explicitly when you use **e**.

The **r** key letter is equivalent to using a write-protect tab on a physical diskette — it prevents you from making any changes to the designated devices. Note that the **r** key letter does not prevent *others* from modifying your files unless you combine it with the **e** key letter.

The order of the **i**, **e**, and **r** key letters is immaterial, but each must be preceded by a comma. These key letters can be used only with devices that ODT-DOS interprets as files — the physical DOS partition, virtual partitions, and virtual floppies.

Specifying Custom Devices

Usage: +a*custom_device*

You, your system administrator, or your ODT-DOS distributor can create device definitions in the file **/etc/dosdev** for hardware devices other than those described in the previous sections. See *Administering ODT-DOS* for information on setting up and using custom hardware devices, including details on **/etc/dosdev**.

Canceling Device Specifications

Usage: **-a**[*device_name*]

The **-a** option prevents access to DOS devices when you use it on the **dos** command line or install it in an application. You can also use **-a** to remove device specifications from options files. The **-a** option can affect access only to those devices you can attach with the **+a** option. You cannot use **-a** to affect access to devices such as the diskette drive that are managed automatically by the ODT-DOS system.

Using -a To Prevent DOS From Accessing Devices

On the **dos** command line, **-a** *device_name* prevents DOS from attaching the specified device even if there is an installed **+a** option in an options file or application that requests that the device be attached. For example, the command:

```
$ dos -ae:
```

starts the DOS environment without allowing access to the DOS partition. It is not necessary to type dos -ae:=/dev/dsk/dos. You do not have to supply the UNIX pathname of the device when you use **-a** with the name of a DOS partition (either physical or virtual) or a virtual floppy.

When installed in a DOS application, the **-a** option automatically prevents access to specified devices or to all devices. **-a** *device_name* prevents access to the specified device unless you override the **-a** option with an explicit **+a** option. For example, the command:

```
$ dosopt -acom1 xtalk
```

prevents CROSSTALK from requesting access to the COM1 port unless you specify **+acom1** on the **dos** command line.

Use the **-a** option without a specific device name to deny access to *all* devices. For example, the command:

```
# dosopt -a xtalk
```

prevents CROSSTALK from accessing *all* DOS devices. Users can override the installed **-a** by using a **+a** option on the **dos** command line.

Installing the **-a** option in an application to deny access to a device is not the same as *removing* the **+a** option. For example, assume you have installed the **+acom1** option in the application CROSSTALK. The command:

```
$ dosopt -acom1 xtalk
```

replaces the **+acom1** request with **-acom1**. Then, when you run CROSSTALK from the UNIX shell, it does not request access to COM1 even if **/etc/dosapp.def** or your home directory **dosapp.def** contain explicit **+acom1** instructions.

Refer to the description of the **+z** option later in this chapter for information on removing options from applications.

Using -a To Remove Defaults From Options Files

You can use a command with the following syntax to remove device specifications from options files:

dosopt -a[*device_name*] *options_file*

For example, the command:

```
$ dosopt -acom2 /usr/dave/dosenv.def
```

removes the **+acom2** specification in **/usr/dave/dosenv.def** but does not affect other **+a** options in that file. Similarly, the command:

```
# dosopt -a /etc/dosapp.def
```

removes *all* device specifications from **/etc/dosapp.def**.

When device specifications are removed from **/etc/dosapp.def** or **/etc/dosenv.def** files, users can still request access to any device by installing **+a** options in their home directory **dosapp.def** or **dosenv.def** files, by installing **+a** options in their own applications, or by using **+a** on the **dos** command line.

Display-Oriented And Stream-Oriented DOS Programs

Usage: ±b

Factory default: -b (display-oriented)

When a DOS program does all its I/O via standard I/O, it is called *stream-oriented.* Most standard DOS commands, such as DIR, SORT, and TREE, are stream-oriented programs. DOS applications that don't write directly to the system screen (many compilers, for example) are also stream-oriented programs.

When a DOS program avoids some or all of the standard I/O mechanisms, it is called *display-oriented.* DOS applications that write directly to the system video memory (including many text-processing, database, and spreadsheet programs, as well as most games) are display-oriented programs.

To properly handle I/O for a DOS program, ODT-DOS must know whether it is stream-oriented or display-oriented. The standard DOS commands in **/usr/dbin** are assigned the appropriate value (in most cases, stream-oriented) when ODT-DOS is initially installed. All other DOS programs are assumed to be display-oriented unless you specify otherwise. You can run any DOS program with the default assignment of display-oriented. To take advantage of the additional flexibility available with stream-oriented programs, though, you should identify these programs as such using the **+b** option. When you identify a stream-oriented DOS program with the **+b** option, you can use UNIX pipe and I/O redirection mechanisms to manipulate the program's input and output.

To determine whether a program is display-oriented or stream-oriented, follow these guidelines:

- Interactive programs are generally display-oriented and should have the default (**-b**) assignment.
- Any program that requires a graphics display adapter is display-oriented.
- Most noninteractive programs (such as compilers) are stream-oriented.

If you are uncertain whether a noninteractive program is really stream-oriented, you can perform the following test. Start the DOS environment

and run the program in question, redirecting the input or output (as appropriate) to a file. For example, to test output redirection for the program APPL, issue these commands:

```
$ dos
C> appl > temp
```

When the program has finished executing, display the contents of the file you redirected to (TEMP in this example). You can do this with the DOS TYPE or UNIX **cat** command:

```
C> type temp
$ cat temp
```

or use a text editor to display the contents of the TEMP file. If the output of your DOS program is correctly captured in this file, the program has stream-oriented output.

To find out whether APPL accepts stream-oriented input, you can test it with a command such as:

```
C> appl < con
```

to test input redirection from the console. If APPL behaves correctly, it accepts stream-oriented input.

NOTE: If you run a program that is really display-oriented with the **+b** option, screen output is lost. To abort the program, send a UNIX interrupt (by default, the DEL key).

Pass Command Directly To DOS Command Interpreter

Usage: **dos +c** *command*

Factory default: None

ODT-DOS by default performs the following actions when you run a DOS command from the UNIX shell:

- Searches your UNIX path for the executable program.
- If the executable program is found, interprets any DOS options that are installed in the file.

The +c option causes the command to be passed directly to the DOS command interpreter, **command.com**. ODT-DOS does not search the UNIX path or interpret any DOS options installed in the application when you use +c.

The +c option is useful with commands that use names (such as DOS device names) that are not meaningful to UNIX. For example, the commands:

```
$ dos +c a:wp
$ dos +bc "dir | sort"
```

run Word Perfect from the diskette drive and sort the output of the DIR command. ODT-DOS runs both commands properly even without the +c option, but it finds the programs more quickly when the +c option is included in the command. Without the +c option, ODT-DOS searches your UNIX search path for files called, literally, "a:wp" or "dir | sort" before passing the command to **command.com**. With the +c option, the command is passed immediately, without searching your UNIX path.

Although +c prevents ODT-DOS from interpreting DOS options installed in the application, options files (**/etc/dosapp.def** or **$HOME/dosapp.def**) are still interpreted. Any options specified on the **dos** command line are also interpreted.

Set Initial Current Drive

Usage: +d[*drive*]

Factory default: +dc

The +d option sets the initial current DOS drive when you enter the DOS environment or run a DOS program. The *drive* parameter can be any drive letter from "a" to "z." For example:

```
$ dos +dd
```

puts you in the DOS environment with drive D: as your current drive. The +d option with no *drive* specification causes the default drive (C:) to be used.

When DOS runs, the drive you specify with +d must exist. If it doesn't, DOS aborts.

Interpret Configuration File

Usage: +e[*file*[,*file* ...]]
-e

Factory default: +e (interpret /config.sys and $HOME/config.sys)

Use the ±e option to tell ODT-DOS whether or not DOS should interpret a configuration file when it runs. By default (+e), DOS interprets two files in succession:

1. The system default **config.sys** file in the root directory (/**config.sys**).

2. Your personal home-directory **config.sys** file.

$HOME/config.sys files do not exist until users create them. If one of these files does not exist, ODT-DOS interprets the file that does exist. When both files exist, ODT-DOS behaves as though the contents of /**config.sys** and the home-directory **config.sys** are all in a single **config.sys** file.

To specify that you want a different file to be interpreted when you run DOS, name the file (by full path name) along with the +e option. For example:

```
$ dos +e/usr/dave/dbin/config.sys
```

runs the DOS environment with the configuration specified in /**usr/dave/dbin/config.sys** instead of the configuration specified in /**config.sys** and **$HOME/config.sys**. The specified file can be any UNIX file — it need not be called "config.sys." It must, however, contain valid **config.sys** commands.

When you specify two or more files, DOS interprets the contents of the files, in order, when it starts. For example:

```
$ dos +e/usr/ldbin/config.sys,/usr/dave/dbin/config.sys
```

No default **config.sys** files are interpreted automatically when a file name appears following +e. Therefore, if you want one or more files interpreted *in addition* to the defaults, you must list all files explicitly:

```
$ dos +e/config.sys,/usr/dave/dbin/config.sys
```

The -e option tells DOS not to interpret any configuration file.

Help Text

Usage: +h

Factory default: None

Use the **dos +h** or **dosopt +h** command to display a help screen that briefly describes all the available options. You can use the **+h** option only on the **dos** or **dosopt** command line. It cannot be installed in an options file or in an application.

Alternative DOS Load Image

Usage: +l[*file*|*directory*]
-l

Factory default: +l (use system default image)

The default DOS images are in files named **mono.img**, **cga.img**, **ega.img**, and **vga.img** in the directory **/usr/lib/merge**. The option +l without a file or directory name means the default image. The option +l*file* uses the specified DOS image file rather than the default DOS image to execute the invoked program. *file* is the full path name of the alternative image, which can have any legal UNIX name.

In the +l*directory* form, *directory* is the directory that contains the DOS image you want to use. When you use this form, ODT-DOS automatically searches the specified directory for an image file that corresponds to your display. The image files in the specified directory must be named **mono.img**, **cga.img**, **ega.img**, or **vga.img**.

When you use EGA or VGA images, the directory containing the image must also contain the file **egarom.img** or **vgarom.img** in addition to **ega.img** or **vga.img**. These files are created automatically when you create the corresponding image file. The procedures for creating DOS images are described in *Administering ODT-DOS*.

It is rarely necessary to store different alternative DOS images and load them with the +l option because ODT-DOS accommodates nearly all configuration changes automatically when DOS is run.

The **-l** option starts a "DOS environment" with no image, booting the virtual machine's BIOS from the reset address f000:fff0. You can use the **-l** option to boot either a physical or virtual diskette. The diskette need not be a DOS diskette as long as it is bootable and is compatible with the virtual personal computer environment you have specified. Many games designed for personal computers, for example, do not run under DOS and need to be booted from the diskette drive using the **-l** option. To use such a game, insert the diskette in the diskette drive and type:

```
$ dos -l
```

If the game is stored in a virtual diskette, you can run it by typing:

```
$ dos +aa:=pathname -l
```

where *pathname* is the name of the virtual diskette. When you run a "DOS" session this way on the console, you can use the UNIX MultiScreen feature to view and work with other concurrent UNIX or DOS sessions. The process booted with the **-l** option, however, typically runs independently of any other UNIX or DOS activity. That is, usually there is no connection between the process invoked with **dos -l** and the shared DOS/UNIX file system, and no interprocess communication between DOS or the UNIX system and the **dos -l** process.

To stop a session started with **dos -l**, press CTRL-ALT-DEL or use the KILL DOS terminal control code described in *Using ODT-DOS*.

The **dosboot** command is a synonym for **dos -l**.

DOS Memory Size

Usage: +m*n*

Factory default: +m640

Whenever you use DOS on the ODT-DOS system, the DOS process requests a specific amount of memory for the environment used by the DOS process. Using the **+m** option, you can request your preferred amount of memory for a DOS process. To specify explicitly the amount of memory you want, use the form **+m*n***, where *n* is an integer between 64 and 640, representing kilobytes. For example:

```
$ dos +m512
```

requests 512K for the DOS environment. The command:

```
$ dos +m256 maps
```

requests 256K and runs the program MAPS.

ODT-DOS overhead consumes a small amount of the memory available to DOS, so the actual amount of memory available to DOS applications is slightly less than the amount of memory that you request.

Run autoexec.bat

Usage: **+p**[*file*]
-p

Factory default: For the DOS environment (**dos**): **+p**
For DOS programs (*program* or **dos** *program*): **-p**

Use the ±**p** option to tell ODT-DOS whether or not DOS should interpret an **autoexec.bat** file when it runs. The defaults are different for the DOS environment and for applications run from the UNIX shell.

The default for the DOS environment (**+p**) is for DOS to interpret two files in succession:

1. The system default **autoexec.bat** file in the root directory (**/autoexec.bat**).
2. Your personal home-directory **autoexec.bat** file.

DOS applications run from the UNIX shell do not run **autoexec.bat** files by default. When you install an application using **dosadmin**, **dosadmin** automatically overrides this default and installs the **+p** option in the application unless you specify otherwise.

The **/autoexec.bat** and **$HOME/autoexec.bat** files do not exist until the system administrator or individual users create them. If one of these files does not exist, ODT-DOS interprets the file that does exist. When both files exist, ODT-DOS behaves as though the contents of both files are all in a single **autoexec.bat** file.

Use **+p** alone in options files, in applications, or on the **dos** command line to cause DOS to interpret **/autoexec.bat** and **$HOME/autoexec.bat**. Use **-p** to prevent interpretation of any **autoexec.bat** files.

In the form **+p***file*, *file* is the name of a single batch file that DOS is to run. The file can have any legal UNIX name and can be anywhere in the UNIX file system. You must specify it by its full UNIX path name.

When you run DOS with the **+p** option in effect, a temporary file **/tmp/ae*nnn*.bat** is created, where *nnn* is the UNIX process ID of the DOS server. This file is normally removed when DOS exits, but if the DOS server is killed, the file is not deleted.

Change DOS Printer Timeout

Usage: +s[*n*]
-s

Factory default: **+s15** (spool to UNIX with a 15-second timeout)

Both in the DOS environment and from the UNIX prompt, DOS printing by default is handled through the UNIX print spooler. The ±s option controls when accumulated DOS printer output is sent to the UNIX spooler. By default, DOS printer output is sent to the UNIX spooler when any one of the following events occurs:

1. The program or command exits. "Exit" means the program finishes and your DOS or UNIX prompt (C> or $) returns. Many commands (such as the DOS PRINT and COPY commands) exit automatically when they are done. Other programs require a specific user action to exit.

2. The amount of time specified by the printer timeout value has elapsed since DOS last printed. The default printer timeout is 15 seconds.

3. DOS receives a BREAK (for example, you press CTRL-C at the DOS prompt).

Use the +s*n* option to change the default 15-second timeout. The variable *n* is a timeout between 5 and 3600 seconds. If *n* is omitted, the default value of 15 seconds is used. The -s option makes the timeout infinite, which prevents DOS printer output from being sent to UNIX until the DOS process exits or you press CTRL-C.

The ±s option has the same effect as the ODT-DOS **printer** command /t option. The ±s option can be used on the UNIX command line or installed in options files and applications. The **printer** command must be used in the DOS environment. Both ±s and the **printer /t** option are effective only when DOS printer output is spooled to UNIX. When you attach a printer directly to DOS using a command such as dos +alp1, the ±s option and the **printer /t** option have no effect.

Translate DOS Switch Characters And Path Separators

Usage: ±t

Factory default: +t (translate switch characters and path separators according to standard ODT-DOS conventions)

By default (+t), ODT-DOS requires UNIX-style switch characters and path separators when you issue DOS commands from the UNIX shell. For example:

```
$ dir /tmp -w
```

Before passing the command to DOS, ODT-DOS automatically translates this command to:

```
dir \tmp /w
```

The -t option prevents ODT-DOS from translating switch characters and path separators on a DOS command line. For example, consider a hypothetical DOS command COMPUTE, which does simple arithmetic calculations like division and subtraction. With the default translation in effect, a command such as:

```
$ compute 8 - 4
```

would be translated to:

```
compute 8 / 4
```

Translation of '-' and '/' in contexts like this, where they are not intended to be interpreted as path separators or switch characters, can result in undesired behavior.

To prevent this translation, use the command:

```
$ dos -t compute 8 - 4
```

You can also use the **dosopt** command to install the **-t** option in COMPUTE so it is effective each time you run the application from the UNIX shell:

```
$ dosopt -t compute
```

The **-t** option does not prevent the UNIX shell from interpreting metacharacters. If you use metacharacters in a DOS command, follow the guidelines listed in Chapter 2.

Dosopt "Verbose" Mode

Usage: dosopt +v [*options_file* | *program*]

Factory default: None (don't display acknowledgements)

Use the +v option with **dosopt** to display the DOS options installed in an application or options file (**dosenv.def** or **dosapp.def**). For example, a typical display in response to the command:

```
$ dosopt +v database
```

is:

```
*-a *-b *+dc *+e *+l *+m640 *-p +s20 *+t +x
*defaults from /usr/dave/dosapp.def.
```

This display shows the values of all DOS options used when DATABASE is run. These values include both:

- Values that have been installed explicitly in DATABASE with the **dosopt** command:

  ```
  $ dosopt +s20 +x database
  ```

- Values that are taken from an options file because they are not explicitly installed in DATABASE.

The values taken from the options file are marked with an asterisk (*), and the options file is listed in the last line of the display. The options file is either the system default options file **/etc/dosapp.def** or the user's home-directory **dosapp.def** file.

You can use the +v option to display the values installed in an options file:

```
$ dosopt +v /usr/dave/dosapp.def
```

You can include the +v option on the same command line as other **dosopt** options. For example:

```
$ dosopt +v +s20 database
```

This command installs the option **+s20** and then displays all options effective for DATABASE, as described above.

The command **dosopt +v** without a program or options file name prints the values that apply to the DOS environment — your home-directory **dosenv.def** file, if you have one, or **/etc/dosenv.def**, if you don't.

Set DOS Break Checking

Usage: ±x

Factory default: -x (break-checking off)

The ±x option sets the conditions under which DOS looks for the DOS break character (CTRL-BREAK or CTRL-C). Under the default condition (-x), DOS checks for the break character at every input or output. When the break-checking option is turned on (+x), DOS checks for the break character at every system call. The effects of +x and -x are identical to the standard DOS commands BREAK ON and BREAK OFF.

If you have a BREAK command in a **config.sys** file that is interpreted when DOS runs, the **config.sys** entry has precedence over the ±x option.

Refer to your DOS documentation for further information on the BREAK command.

dosopt Null Option

Usage: dosopt +y *program*

Factory default: None

Use the **dosopt +y** option to create a null DOS options record in a DOS application. The +y option is only necessary for DOS .COM files that are

to be executed from the UNIX shell and that have not already had DOS options installed using the **dosopt** command. The options record that +y creates has no effect except that it allows ODT-DOS to identify a DOS .COM file as a DOS program when you run it from the UNIX shell. For example, assume you want to run the DOS CALC.COM program from the UNIX shell. Install the +y option by typing:

```
$ dosopt +y calc.com
```

The standard MS-DOS .COM files distributed with ODT-DOS already have appropriate **dosopt** records, and you don't need to use **dosopt** to change the records unless you want to change the default behavior of these commands. When you use **dosadmin** to install applications as described in *Administering ODT-DOS*, it automatically installs the +y option in .COM files. You need to use the +y option explicitly if you create or install new .COM files without using **dosadmin**.

Remove Assigned Options

Usage: **dosopt** +z*option program*
dosopt +**Z** *program*

Factory default: None

Use the +z option to remove an option previously assigned to a DOS application with **dosopt**. For example, you might use **dosopt** to identify the DOS program WORK as a stream-oriented program requiring 512K bytes of memory and using the DOS configuration file **/usr/dave/config.new**:

```
$ dosopt +b +m512 +e/usr/dave/config.new work.exe
```

Values other than those assigned explicitly to WORK come from an options file—either **/etc/dosapp.def** or **$HOME/dosapp.def**. The command:

```
$ dosopt +zb +ze work.exe
```

removes the +**b** and +**e** options from WORK and allows the option values installed in the relevant options file to take effect. Each +z requires one and only one option letter. If you want to remove more than one option, use +z once for each option you are removing, as illustrated above. The command dosopt +za *application* removes all ±**a** options from *application* if there are multiple ±**a** options installed at the time you use the +z

option. The command dosopt +z*adevice application* removes only the +a*device* option without affecting any other ±a options that might exist in *application.*

The +Z option removes *all* installed options from DOS applications. For example, the command:

```
$ dosopt +Z work.exe
```

removes the complete record of assigned options from WORK and allows all options in the relevant options file to take effect.

You cannot use the +z or +Z option on options files (**/etc/dosenv.def**, **$HOME/dosenv.def**, **/etc/dosapp.def**, or **$HOME/dosapp.def**).

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X