MacDOS™

User's Guide

Version 2.0

Rainbow Hill

MacDOS is a Trade Mark (reg. pend.) of Rainbow Hill Pty Ltd.

Disclaimers and Limited Warranty
MacDOS is provided "as is" without warranty of any kind. Rainbow Hill Pty Ltd expressly disclaims all implied warranties, including but not limited to the warranties of merchantability and fitness for a particular purpose. Furthermore, Rainbow Hill Pty Ltd does not guarantee the correctness, accuracy, and reliability of MacDOS and the accompanying written material. By buying this product, you have assumed responsibility of all consequences of its usage, whether due to user error or software defects.

Under no circumstances, Rainbow Hill Pty Ltd, its developers, directors, and associates will be liable to you for damages to your data or business caused directly or indirectly by the usage of MacDOS.

Rainbow Hill Pty Ltd warrants to the original purchaser of MacDOS that the diskette is free from defects for a period of 90 days from the date of purchase. Defective diskettes returned within 90 days from the date of purchase will be replaced without charge.

Software Licence
Licence is granted to the original purchaser of MacDOS for exclusive use by a single person. MacDOS may be installed on more than one computer provided they are all used exclusively by the same person.

The restrictions specified in this licence only apply to fully functional copies of MacDOS. Demo versions of MacDOS and associated ReadMe files may be copied and freely distributed, provided they are kept together and not altered or modified in any way.

Printed in Australia by Optima Press, East Perth.
First printing: September 1994 .

Section 1

Welcome to MacDOS

1 Welcome to MacDOS

Why MacDOS ?

MacDOS satisfies the need of many Macintosh users to be able to avoid repeating particular sequences of mouse operations when working with files.

Normally, you handle documents on the Macintosh desktop through the Finder. It is the Finder that creates the desktop metaphor by drawing icons and responding to the mouse. Together, the Macintosh Operating System (MacOS) and the Finder have made computers accessible to all those who were intimidated by technical jargon and arcane commands.

Nevertheless, the ease of use of a Graphic User Interface (GUI) does not come free of charge: operations which require brief commands on another computer need sometimes an almost interminable repetition of clicking and dragging on the Mac.

People who have never used any other computer before do not realise it, but others, who have worked with DOS or UNIX, immediately see that limitation. This is partly why many computer professionals have seen the Macintosh more as a toy rather than as a system capable of doing *serious* work.

For the past few years, hundreds of software developers have worked hard at providing a Macintosh-like interface on every conceivable computer. The developers of Macintosh software, on the other hand, have found it difficult to respond to those potential users who would like to have a character-based interface to the MacOS ***beside*** the GUI provided by the Finder.

MacDOS goes against the current tide and satisfies the need of Macintosh power-users by providing a DOS-like interface on the Mac.

Introduction to MacDOS for DOS Users

Although MacDOS provides a DOS-like user interface, it is not DOS. Therefore, you will find that some of the commands do not behave exactly like in DOS. Nevertheless, 80% or more of what most people do with MS DOS® 5.0 is available in MacDOS.

Most importantly, MacDOS is safer than DOS. That is, you will never type a harmless DOS-command and find out that MacDOS responds to it by destroying your data!

Most commands accept full paths as arguments. Their format is like in DOS, but in MacDOS we have replaced the concept of file extension with the more powerful mechanism available on the Mac of creators and file-types. With most commands, you will be able to complement or replace file wildcarding like `*.txt` and `*.exe` with type switches like `/T=TEXT` and `/T=APPL` .

The length of filenames and directories can of course exceed the limit of eight characters imposed by DOS. You can also include spaces and other non-alphanumeric characters in names, provided you enclose the names between double quotes (with the system variable QUOTE you can also change the character used for quoting).

Another change that we made in order to take advantage of specific Macintosh features was the replacement of drive IDs with volume IDs. To avoid confusion, we have identified volumes with numbers rather than letters, but the difference is more than cosmetic: the MacOS assigns IDs to

volumes in the order in which they are mounted and frees the IDs when the corresponding volumes are dismounted. A volume ID identifies a particular floppy, rather than the floppy drive. Therefore, your startup disk partition or floppy is *always* 1 and the volume ID assigned to a newly inserted floppy depends on what volumes you have currently mounted on your desktop.

This might sound a bit confusing, but you will find it natural as you get used to the Mac environment. To help you when importing batch programs from DOS, MacDOS provides the command SUBSTVOL which assigns a letter to a particular volume. This allows you to assign the appropriate letters to the necessary volume IDs before executing DOS batch files within MacDOS.

Similarly to what happens when you boot DOS, MacDOS executes the file `autoexec.bat` found in the folder where the application resides (not where you place aliases of MacDOS). There is no MacDOS-equivalent of `config.sys`, though.

The Command Reference Section provides detailed descriptions of the differences between MacDOS commands and their DOS counterparts.

Introduction to MacDOS for Mac Users

This section is only going to explain very basic concepts of MacDOS' character based interface. Also refer to the DOS Glossary Section for an explanation of the terminology used in this manual.

To understand MacDOS you have first of all to think that the operations possible with files and folders do not depend on the way in which you direct the Operating System to perform them. Moreover, the differences between the *Macway* and the *MacDOSway* are pretty consistent across the operations. Therefore, once you will have discovered how to perform, say, a file copy, you will be close to understanding how to perform file renaming, file listing, etc.

When MacDOS starts, it opens a window, displays a prompt to tell you that it is ready, and waits for you to type a command.

There are many commands that you can type, but they all consist of a verb (the *command name*) possibly followed by some other words (the *command parameters*). The command name tells MacDOS what to do and the command parameters tell MacDOS on what it should operate.

To type a command you just enter its name and parameters from the keyboard. You then press the *return* key to tell MacDOS that the command is complete and can be executed. MacDOS executes the command and then displays a new prompt to tell you that it is ready to accept more commands.

For example, **`TYPE filename`** lists the text file named **`filename`** on the screen. Similarly, **`DEL oldfile`** removes the file named **`oldfile`** from your system, as if you had trashed it and then directed the Finder to empty the Trash. **`RENAME oldname newname`** changes the name of a file from **`oldname`** to **`newname`**, while **`COPY oldname newname`** makes a copy of the file named **`oldname`** and names it **`newname`**.

You can modify the operation of a command by typing some special arguments called *options* or *switches*. For example, you might like that a COPY command verifies that the copy of the file is identical to the original. To achieve that result, type **`/V`** before hitting the return key.

The Finder mostly operates on files or folders which are displayed in the front window. A similar concept applies to MacDOS, where most commands operate on the *default directory* (the term directory is a synonym of folder). When MacDOS is launched, the default directory is the

folder which contains the icon of MacDOS itself, but you can change it with the command **CHDIR**.

You can also list the contents of a folder with the command **DIR**, and there are also commands to create, copy, and trash folders.

If you want to operate on a file which is NOT in the default directory, you can specify the file by prepending to its name a string which identifies where the file is. Such a string is called *path* and consists of the name of all the folders that you would have to open to reach the file. Each folder name is followed by a backslash. For example, let's suppose that the default directory contains a folder called **myFolder** which in turn contains a text file called **myFile**. If you want to print **myFile**, you can either change the default directory to **myFolder** with the command **CHDIR myFolder** and then print the file with the command **PRINT myFile** or do the printing without changing the default directory by typing the command **PRINT myFolder\myFile** .

One of the big advantages of MacDOS over the Finder is that you can use *wildcards* to identify several files simultaneously. For example, **DEL p*** deletes all files whose name begins with the letter **p** (or **P**, as the Mac OS is not case sensitive). Also, **DEL x???** deletes all files whose name begins with **x** and is exactly four characters long.

To interrupt the execution of a command, you can type the standard command-dot key combination.

If there are particular sequences of commands which you need to use over and over again, you can write the commands in a text file and then type the name of the file at the MacDOS prompt. In this way you can effectively extend the standard commands available in MacDOS with your own procedures. Such files are called batch programs.

What do You Need and What is in the Package ?

MacDOS uses 500kByte of RAM and requires System 6.0.8 or greater. Moreover, it requires System 7.0 to be able to launch other Macintosh applications.

MacDOS itself fits into 240kb of disk space, but you should reserve additional space if you want to keep on-line batches, filter extensions, README files, and this User's Guide.

The MacDOS package consists of this manual and a 800kByte 3.5" floppy disk. The floppy contains a self-extracting archive with the MacDOS application, examples of batch files, examples of filter extensions, a shell project to build your own extensions, this User's Guide, and various `README` files.

How to use this Manual

This manual should provide all the information you need to understand MacDOS and to use it efficiently.

Beside the section of general introduction that you are reading right now, you will find:

Section 2: The Environment

How you interact with MacDOS.

If you have no experience with DOS, you should definitely read this section. It will also introduce you to some terminology commonly used in DOS.

If you know DOS, you can skip most of the section but have a look at the parts where the error mechanism is introduced and the MacDOS window explained.

Section 3: Data on the Mac

How the Macintosh organises data and what information visible on the desktop is accessible via MacDOS.

If you have ever looked into *Inside Macintosh*, you can probably skip this section.

Sections 4 to 12: MacDOS Functionality

What MacDOS can do, organised on the basis of functional areas.

The functionality of all MacDOS commands is explained in a coherent way and without letting the details cloud the big picture. You should definitely browse through these sections if you have no experience in DOS.

If you have DOS experience, you should still look at the commands which are not available in DOS and find out how they fit into the general strategy.

Section 13: Command Reference

One chapter on each command, with all options explained.

Several examples are provided, as well as possible causes of commonly occurring errors. For commands which exist in DOS, the differences between the two implementations are listed.

You should refer to this section before using a command with which you are not familiar.

Appendix A: Error Codes

Each possible error code returned by MacDOS is explained, with some indications of the possible causes.

Appendix B: Extension Programming

How to write filter applications that accept piped MacDOS commands.

Appendix C: DOS Glossary

DOS concepts are explained in terms of Macintosh concepts.

Appendix D: Command Index

The full list of references to MacDOS commands.

Conventions

Full commands, command names, and arguments appear in the `Courier font`.

When needed for clarity, commands and part of commands embedded in normal text are in **`bold face`**.

Section 2

The Environment

2 The Environment

The Macintosh Finder allows you to perform a series of operations by clicking and dragging icons and windows with your mouse, while MacDOS expects you to type commands on the keyboard.

MacDOS does not normally display a cursor changing from a clock to an arrow to tell you when an operation has been completed. Instead, whenever MacDOS completes the execution of a command and is ready to accept the next one, it displays a prompting string at the beginning of the bottom line of the screen. That line is called *Command Line*. With commands affecting several files and folders at a time, MacDOS also displays the name of each item before it begins operating on it.

A command is like an order in English: a verb in the imperative form, possibly followed by one or more objects and some qualifiers. Commands are usually case insensitive.

The verb is called *Command Name*, the objects *Parameters*, and the qualifiers *Switches* or *Options*.

After completing the typing of the command, you direct MacDOS to start its execution by pressing the RETURN key.

Command Name

The command name must be the first word of the command and is never case sensitive. Also, command names provided by MacDOS never contain spaces or other special characters.

Parameters

Most commands need one or more parameters to identify the objects they are requested to operate on. An example of commands which do not require any additional information is `CLS`, used to clear the MacDOS window.

Many commands accept different numbers of parameters and automatically "fill in" the missing parameters with default values. For example, the command `DIR` displays lists of files in the MacDOS window and expects one parameter which identifies the files to be listed. Nevertheless, you can just type `DIR` without parameters to list the contents of the current folder.

Parameters are sometimes case sensitive. For example, when you rename a file you have to type the new name exactly as you want it, with upper and lower case characters.

The order of parameters is almost always fixed. For example, if you want to rename a file, you have to type the old name first and the new name second. A case in which the order is not important is when you copy several files with a single `COPY` command. In that case, you type the filenames one after the other separated by commata. MacDOS will copy the files in the order in which you specify them, but this should usually have no practical consequences.

Switches

Switches always start with a slash (the character '`/`'). This is how MacDOS distinguishes them from parameters. The slash is always followed by a character which identifies the particular switch, and possibly other characters which select different ways of operation. For example, you

obtain a sorted list of files by adding the switch /O to the command DIR. You can also specify ways of sorting by adding other characters to the switch: DIR/ODS displays a list of files sorted on the basis of their last upDate and Size (update first), while DIR/OSD lists the same files but giving priority to their size rather than the date and time of last update.

As switches are always used to request optional features, they are never mandatory. Nevertheless, very often you will need to use one or more switches. For example, to avoid that a long list of files disappears off the top of the MacDOS window, you can type DIR/P instead of simply DIR.

As with parameters, switches are seldom case sensitive. One case sensitive switch is /T, used to select a file-type. The 'T' of the switch can be in lower or upper case, but the file type must be correct: /t=TEXT is different from /t=TeXT because the two types 'TEXT' and 'TeXT' are seen as different by the Mac Operating System.

The order and position of switches within a command are in most cases irrelevant. In fact, MacDOS extracts and analyses all the switches before looking at the parameters.

One switch accepted by **all** commands is /?. When you type this switch in any position, MacDOS ignores all other parameters and switches and displays a brief description of the command.

Redirection and Piping

Most commands produce some output messages, either because that is their purpose, or because they provide information on their progress. For example, the purpose of the command DIR is precisely to produce a list of filenames. On the other hand, the COPY command lists the names of the files being copied.

By default, MacDOS displays these messages in the MacDOS window but you can decide to re-direct them to a text file. To do so, you must add to the command a "greater than" character followed by the name of the text file. For example, DIR>MyFile lists the contents of the current folder in the text file MyFile rather than on the screen.

When you redirect the output, MacDOS produces files which can be directly opened with any text editor or word processor.

Note that error and warning messages are never redirected to files. MacDOS always displays them on the screen. Also, if you attempt to redirect the output of commands which have nothing to redirect, MacDOS displays a warning message, ignores the redirection, and executes the command as if you had not attempted a redirection.

You can append the output of a command to an existing file, rather than create a new one, by typing two "greater than" signs instead of one (eg. DIR>>MyFile).

The commands which accept output redirection are:
ALARM, CALL, CD, CHDIR, CLOSE, CONFIRM, COPY, DATE, DIR, EJECT, EXIT, HELP, LOG, MD, MKDIR, MEM, MORE, NEXT, ONERROR, OPEN, PATH, PRINT, PROMPT, READ, RD, REM, REN, RENAME, RESTART, RMDIR, SERIAL, SHOW, SHUTDOWN, SUBSTVOL, TIME, TREE, TYPE, VER, VERIFY, VOL, WRITE, XCOPY .

Some of these commands do not actually produce any output. Therefore, their acceptance of output redirection is simply an indication that they will not report a warning message if you do attempt to redirect their output.

Beside redirecting the output produced by a command, you can also redirect the input from which the command MORE accepts data. You do this by appending to the command line a

"less than" character followed by the name of the text file containing the data. This is useful in order to break down long displays into pages. For example, `MORE<MyFile` displays the text file `MyFile` one page at a time and prompts you to continue or interrupt the display at the end of each page.

Input redirection is also possible with MacDOS extensions (ie. MacDOS filter applications). Please refer to the appropriate section for the details.

Instead of redirecting the output of a command to a file and then use that file as input to `MORE` or to a filter application, you can combine the two commands and achieve the same result in a more efficient way. This mechanism is called *piping*. You just need to type the commands one after the other separated by a vertical bar (the character '`|`', called *pipe*). For example, if the on-line help displayed by the command `COPY/?` is too long to be contained in the MacDOS window, you can type `COPY/?|MORE` and MacDOS will wait for you at the end of each page.

Pipes are only relevant for:
- Internal commands which allow output redirection.
- The command `MORE` (ie. the only internal command which accepts input redirection).
- MacDOS filter applications.

The command `FOR` is a special case, because it passes its output or input redirection to the command which follows the `DO` keyword. Therefore, although `FOR` itself does not accept redirection, a command line beginning with `FOR` can contain I/O redirection. Only single-line `FOR`s accept I/O redirection. Moreover, input redirection and piping is only possible in conjuction with `MORE` .

What follows is a complete list of the rules that govern redirection and piping in the MacDOS environment. In order to fully understand all its details, you will probably need to come back to this list after reading some other sections of this User's Guide.
- In a single command, you can only redirect input and output once, and their order is irrelevant. If the line includes pipes, the input redirection applies to the first command and the output redirection to the last one, regardless of where the redirections actually appear within the line.
- MacDOS displays a warning message whenever the user applies input or output redirection to a command that does not support it.
- Most internal commands accept output redirection to file, but only `MORE` accepts input redirection. Single-line `FOR`s accept input redirection when their `DO` is followed by `MORE`, in which case the redirection is considered to be referring to the `MORE` rather than to the `FOR` itself. Single-line `FOR`s accept output redirection, but multi-line `FOR`s don't.
- Batch programs accept output redirection to file but not input redirection.
- Applications and AppleScripts do not accept any I/O redirection.
- MacDOS extensions accept both input and output redirections. Therefore, a chain of filters can obtain its input from a disk file rather than from a command.
- `MORE` can only appear at the end of a command line (this includes the case in which there is nothing else and `MORE` is on its own).
- A command line containing pipes can only begin with an internal command which supports output redirection (but excluding `MORE`), or with a filter. If it begins with a filter, the command line must include input redirection.
- The command `FOR` only supports a pipe in the following case:

  ```
  FOR %var [/L] IN (set) DO internal-com-with-params | MORE
  ```

  That is, in the single-line format and with a single pipe used for `MORE`.

Errors and Warnings

MacDOS distinguishes between serious conditions which cause a command to be rejected or aborted (errors) and others which should not occur but can be tolerated (warnings).

In any case, MacDOS displays a message to inform you that an abnormal condition has been detected.

Most errors are due to the presence in a command of invalid parameters and switches, or the request to operate on something that does not exist.

Warnings are usually caused by the attempt of redirecting I/O of a command which does not support redirection. You also get a warning when you attempt to use a DOS switch which is not supported by MacDOS.

MacDOS normally reports errors and warnings by displaying a short message. You can use the `ALARM ON` command to have all error and warning conditions reported via an alert dialog box. MacDOS then pauses whenever it detects a problem and provides in the alert box some information which is not normally displayed.

Refer to Appendix A for an indication of the possible causes of error and warning messages and some advice on possible remedies.

Menus and Window

The MacDOS window accepts commands and provides space for displaying command outputs. This is why throughout this manual it is also referred to as *the console window*. You can change the character size of the console window through the system variable FONTSIZE.

MacDOS supports the standard mouse operations to work with text: cut, copy, paste, and clear. You can select text with the mouse and then select the appropriate item of the Edit menu or type the keyboard shortcut to perform the operation you need.

This frees you from having to type long file and folder names, as you can copy names from a directory list and paste them into the command line. To make such an operation easier, MacDOS selects a whole quoted string (quote characters included) when you double click between a pair of matching quotes.

Note that you cannot paste carriage return characters into the command line, because MacDOS interprets a carriage return as an order to execute a command.

Typed characters are only accepted within the command line. If the selection is entirely outside the command line, the insertion point is moved to the end of the command line before showing the character typed. If, on the other hand, a selection covers part of a command, the selection is reduced to that part before replacing it with the character typed (ie. the left edge of the selection is moved to the first character of the command line).

Only the command line can be altered. Therefore, outside of the command line:

- "Cut Text" behaves like "Copy Text".
- "Paste Text" moves to the end of the command line before pasting, effectively attaching the content of the clipboard to the end of the command.
- "Clear Text" has no effect.

If the selection includes a part of the command line, "Copy Text", "Paste Text", and "Clear Text" behave as if the selection only consisted of the part which is within the command line. That is, only the part of selection which is within the command line is cut, replaced, or cleared.

Up and down arrows scroll through the previous and following commands. This possibility is particularly useful when you mistype a character of a long command, because you can "recall" the command by typing an up-arrow. You can then correct the mistake instead of having to retype the whole lot.

Left and right arrows extend selections when they are shifted and move to the beginning and the end of the command line when they are pressed together with the option key.

MacDOS also responds to the "Select cmd line" item of the Edit menu by highlighting and selecting whatever you have already typed after the current prompt. Similarly, "Select All" highlights the contents of the whole console window.

The two items "Previous Cmd" and "Next Cmd" of the Edit menu (and their corresponding shortcuts cmd-dash and cmd-equal) behave like the up and down arrows.

The console window accepts vertical scrolling and resizing via the standard vertical sidebar and boxes. The buffer is guaranteed to store at least the last 24000 characters typed or displayed.

You can clear the console window with the command `CLS` (which stands for CLear Screen).

To print the contents of the console window, select the "Print console..." item of the File menu or simply type the command `PRINT` . You can also set the format of the printed page by clicking on the item "Page Setup..." of the File menu.

To close the console window and terminate MacDOS, you can:
- Click on the "go away" box in the top left corner of the window.
- Select either the Close or Quit item of the File menu.
- Type cmd-Q.
- Type the command `EXIT` .

If you do not remember how to display MacDOS on-line help, select the "Help..." item of the apple menu. It will tell you how to use the command `HELP` .

If some colleagues or friends would like to try MacDOS, you can create a demo version of MacDOS by selecting the "Make DEMO..." item of the File menu. MacDOS will then duplicate the MacDOS application file and disable the appropriate functionality.

Control Codes

You can abort the execution of most commands by typing cntl-C or cmd-dot. MacDOS achieves this by monitoring the keyboard at times when the current command can be safely interrupted.

Two additional control codes are available: cntl-S and cntl-Q. They pause and resume long listings on the monitor screen. For example, instead of using the command `MORE` to list a text file one page at a time, you might choose to use the command `TYPE` and press cntl-S when you spot something which you want to read. You can then type cntl-Q to resume the listing or cntl-C to abort it.

Tabbing

You can use the TAB key to speed up the typing of long file and folder names and the shift-TAB key combination to enter frequently used strings.

TAB

When you press the TAB key, MacDOS interprets the preceding characters as a part of a file or folder **name** (with or without path) and attempts to **complete** it. To determine what portion of

the command line to use, MacDOS examines the preceding characters in reversed order and stops when it finds a **quote** or a **space**.

It then scans the current folder looking for matching names of files or subfolders:

- If MacDOS does not find any match, it redisplays the command unchanged.
- If MacDOS finds a single match, it completes the name within the command line and redisplays the command.
- If MacDOS finds several matches, it lists them before redisplaying the command unchanged. In any case, MacDOS truncates the list of matches if it occupies more than half of the console window. If the matches have in common more characters than those typed by the user, MacDOS extends the partial name before redisplaying the command. This mechanism is particularly useful when navigating hierarchies of folders, because it lets you type the minimum number of characters needed to identify any particular item.

Whenever MacDOS completes or extends a name and finds that the name was not quoted, it inserts the opening quote character.

Shift-TAB

When you type TAB while pressing the SHIFT key, MacDOS interprets the preceding characters as part of an **abbreviation** and attempts to **replace** it with a full string.

For a description of this feature, please refer to the "Abbreviations" subsection of the "Customising MacDOS" section.

Section 3

Data on the Mac

3 Data on the Mac

The purpose of this section of the User's Guide is to introduce you to how the Macintosh Operating System organises data. Nevertheless, many functions are left out because they have no visible effect on MacDOS (although MacDOS itself is aware of them!). To have a full and detailed description of the Macintosh File Manager, you should refer to Apple's Inside Macintosh volume series.

Volumes and Drives

Inside Macintosh Vol II defines a **volume** as a *piece of storage medium [...] formatted to contain files*. Removable media like floppies are volumes.

When you introduce a floppy into the floppy drive, the Mac OS assigns to it a Volume Reference Number and forms a volume record to store information about the floppy. The volume is then said to be "mounted" and "on-line".

When the floppy is trashed, the Mac OS frees the Volume Reference Number and the memory containing the volume record. The volume is then said to be "unmounted" and the Mac OS completely forgets the floppy. Only volumes which are on-line are fully accessible by applications.

When you eject the floppy (ie. when you type cmd-E and the icon becomes grey), the Mac OS releases most of the information concerning the floppy but retains the Volume Reference Number. The volume is then said to be "mounted" but "off-line". When an application tries to access a volume which is off-line, the Mac OS asks you to re-insert the floppy, so that it can be placed back on-line.

If you boot a Mac from a hard disk, insert one floppy, eject it with cmd-E, and then insert a second floppy, the Mac OS assigns VRefNum 1 to the startup disk, 2 to the first floppy, and 3 to the second.

If you then unmount volume 2 by trashing the icon of the first floppy, the Mac OS frees VRefNum 2 but does **not** re-arrange the numbers. Therefore, the second floppy remains volume 3.

Finally, if you eject with cmd-E the second floppy instead of trashing it and then insert another floppy, the Mac OS re-uses VRefNum 2. In other words, the same Volume Reference Number can be assigned to different floppies at different times.

A **drive** is the physical device on which you can mount a volume (eg. a floppy disk drive). At startup, the Mac OS assigns Drive Numbers to all drives and never changes them thereafter. For instance, the internal floppy drive is always drive 1, the external floppy drive is drive 2, and so on. Most Operating Systems (including DOS) operate on Drive Identifiers (the A:, C:, etc of DOS, which are effectively equivalent to the Macintosh Drive Numbers).

Beside never changing, a Drive Number also remains valid regardless of whether a floppy is ever mounted on the drive or not.

Especially if you are an experienced DOS user, it is important that you fully understand the differences between the two concepts of "volume" and "drive".

Folders

A **folder** is a collection of files and other folders.

You can use folders as if they really *contained* files, and the desktop metaphor reinforces this view. Nevertheless, folders are more like a telephone directory, in that they contain the information necessary to find the files (ie. the people) rather than the files themselves. This is why it is so quick to move files from one folder to another: the files remain where they are and only a short directory entry is moved.

This misconception of folders actually containing files leads to the concept of a "folder size". In reality, folders have no size of their own. If you work with System 7.0, try this: empty the Trash, create an empty folder, trash it, and then empty the Trash once more. The Mac OS will display a dialog box saying: *The Trash contains 1 item. It uses zero K of disk space ...*

A folder name can contain up to 31 characters and most ASCII characters are accepted.

Files

Again from Inside Macintosh Vol II: *A file is a finite sequence of numbered bytes*.

Macintosh files can consist of two parts called **forks**: the data fork and the resource fork. These two parts have different functions and are usually accessed separately.

The **resource fork** is the structured part of a file and contains objects like menus, fonts, icons, pictures, and code. All application files have a resource fork. In fact, most applications have no data fork at all.

The **data fork** is the unstructured part of a file and contains data used by applications, like text, database records, and format preferences. Many document files contain a data fork but no resource fork.

File Names

A file name can contain up to 31 characters and most ASCII characters are accepted.

File Updates

For each file, the Mac OS remembers the date and time of its last update. Files are only considered updated when their content is changed. Therefore, operations like renaming, changing of attributes, moving, and copying do not modify the date and time of the last update.

File Finder Attributes

The Mac OS keeps data concerning the appearance of files on the desktop. This information is stored in records called "Finder Info"s, and each file has its own FInfo record. The fields which you need to know in order to work with MacDOS are:

File Creator

The Creator of an application identifies the application itself. The Creator of a document identifies the application which created it. In practical terms, the Creator of a document determines which application is launched when you double click on the document icon.

All creators are registered with Apple, so that they are unique. They consist of four characters and are case sensitive. For instance, MacDOS' creator is `'mDOS'` and Teach Text's is `'ttxt'`. The Creator `'????'` is often used to identify files which were created by unregistered applications.

File Type

The Type of a file identifies its function. It consists of four characters and is case sensitive.

Apple has defined some standard types which many applications use. For example: plain ASCII files are of type `'TEXT'` and applications are of type `'APPL'`. In addition, many applications define special file types for their own use.

The Finder uses Creator and File Type to determine what icon to display for a particular file.

Creators & Types Relevant for MacDOS

- Creator `'mDOS'`; type `'APPL'`
  The MacDOS application file.
- Creator `'mFLR'`; type `'APPL'`
  Applications that can operate as MacDOS extensions and interact with MacDOS through pipes.
- Creator `'ToyS'`; type `'TEXT'`
  Text AppleScripts that can be executed.
- Creator `'mDOS'`; type `'TEXT'`
  Batch files that can automatically launch MacDOS when double-clicked.
- Creator `'ttxt'`; type `'TEXT'`
  Text files created by TeachText. By default, MacDOS creates all text files with this creator.
- Creator `'hhgg'`; type `'INIT'`
  File sharing extension, to be switched off before quitting the Finder.
- Creator `'MACS'`
  The creator of the Finder.
- Type `'APPL'`
  Generic applications that can be launched.
- Type `'osas'`
  Compiled AppleScripts than can be executed.
- Type `'TEXT'`
  Generic text files used for I/O redirection or to be executed as batch files from within MacDOS.

Hidden Flag

Within the FInfo record, a flag determines whether the icon of the file is to be displayed on the desktop or not. This flag is used to protect system files and other files which should not be accessed with the mouse during normal operations.

Usually, you can access the hidden flag through special applications like ResEdit. MacDOS lets you list hidden files and work with them. It also lets you easily toggle the hidden flag with the command `REN/H`.

Aliases

Aliases are files which store the location of other items like files, folders, and floppies. Apple introduced them with System 7.0 .

In most cases, you can use an alias as if it were the corresponding target, because the Mac OS hides the connection and automatically refers to the target.

The way in which MacDOS handles aliases can be summarised as follows:

- The `DIR` command, which lists the contents of a folder, handles aliases as if they were normal files except when the user applies the command to a single aliased folder. In that case, `DIR` resolves the alias and lists the contents of the target folder. In lists of filenames,

aliases produce double entries which provide information on both the alias itself and the target.

- Commands which operate on files always handle aliases like files, so that aliases can be renamed, copied, and deleted like normal files, regardless of whether they have as target a file or a folder.

- Commands which operate on folders fail when applied to an aliased folder, with one exception: `CHDIR` applied to an aliased folder resolves the alias and sets the default directory to be the target folder.

Section 4

Working with Volumes

4 Working with Volumes

This section describes the MacDOS commands which let you work with volumes. Refer to the section "Data on the Mac" for an introduction to the concept of "volume".

Many MacDOS commands let you specify a volume as part of parameters which identify files and folders. In those cases, if you do not specify a particular volume, MacDOS assumes that the files or folders are on what is called the "current volume".

When you launch MacDOS, it automatically sets the current volume to the volume that contains the MacDOS application. If you never need to refer to other volumes (like floppies and other hard disks), you can work with MacDOS without ever having to specify a volume at all.

MacDOS identifies volumes through volume IDs. You can assign new IDs to volumes, display data of particular volumes, and operate on different volumes.

Volume IDs

The Mac OS uses a number called Volume Reference Number to identify a volume. MacDOS extends this concept to that more general one of volume ID by including user-assigned letters.

You assign literal volume IDs to volumes with the command `SUBSTVOL`, which operates on volumes regardless of whether they are mounted or not. Therefore, you can set up your literal volume IDs before mounting the corresponding volumes. Nevertheless, you will not be able to use the new IDs until you actually mount the volumes they identify.

If you want to know all current assignments, type `SUBSTVOL` without parameters.

Whenever a volume ID could be confused with something else, you need to append a colon. For instance, if you type `"DIR c"`, MacDOS attempts to display the directory entry of the file named '`c`'. If you want to list the content of volume '`C`', you need to type `"DIR c:"`. If you take the habit of always typing the colon, you cannot ever be wrong.

The startup volume is always identified by volume ID 1.

Displaying Volumes

If you want to know the ID and name of the current volume, type the command `VOL` and MacDOS will provide the information. `VOL` also notifies you when a volume is locked.

If you want to have information on a volume other than the current one, type `VOL` followed by the volume ID of the volume you are interested in. Also the commands `DIR` and `CD` display ID and volume name (together with other information).

If you execute the command `PROMPT` with the options `$N` or `$P`, MacDOS will include the ID of the current volume in the command prompt.

Changing the Current Volume

When MacDOS is launched, the current volume is automatically set to the volume which contains the MacDOS application file. This is true regardless of whether you launch MacDOS by

double clicking on a text file with creator 'mDOS', on an alias of MacDOS, or on the application file itself.

To make a different volume current (in jargon: to *attach* to a different volume), just type the new volume ID, a colon, and RETURN. MacDOS interprets a volume ID on its own as a request to change the current volume, checks whether the requested volume is mounted, and attaches to it.

You can attach to any mounted volume, regardless of whether it is on-line or not. Only when MacDOS needs to access the volume it checks whether the volume is off-line and asks you to put it back on-line (eg. to re-introduce the floppy).

MacDOS commands recognise volumes through their IDs, but the requests to put a volume back on-line refer to volume names. If you realise that the name is different from what you expected and want to go back to the MacDOS prompt, type cmd-dot or cntl-C.

Ejecting volumes

You can dismount removable volumes (eg. floppies) with the command EJECT followed by the volume ID. If you just want to put the volume off-line but keep its icon on the desktop, type the same command with the switch /E.

Section 5

Working with Folders

5 Working with Folders

This section describes the MacDOS commands which let you work with folders. Refer to the section "Data on the Mac" for an introduction to the concept of "folder".

Several MacDOS commands operate on folders. To tell MacDOS where to look for them, you specify the folder which contains them. If you do not specify any containing folder, MacDOS operates within what is called the "current folder".

Identifying a Folder

A folder is identified through its name and its location. The name alone is not enough because there could be several folders in the system with the same name. The location is nothing more than the identification of the folder where the folder in question is to be found. This only shifts the problem of identification "up" (or "out") one level, but it works if you repeat it until you arrive to the top (or outermost) folder on the volume. Such a folder is unique for each volume and is referred to as "the root" of the volume.

To put it more clearly, in order to identify a folder you must provide:

- Identification of the volume where the folder resides (ie. the volume ID).
- The name of all folders, one inside the other, which you need to open in order to "reach" the folder you are looking for. You start from the outermost folder on the volume (ie. the root) which is unique on the volume.
- The name of the folder itself.

Such an identification in MacDOS looks something like this:

```
1:\fold1\aDeeperFold\still deeper\myFold
```

where:

- "1" is the volume ID.
- The first backslash stands for the root, which does not need to be named because it is unique on the volume.
- "fold1", "aDeeperFold", and "still deeper" are the names of the three folders which are between the root and the folder you want to identify. Note that the backslashes delimit the names but do not form part of them.
- "myFold" is the name of the folder you are looking for.

Folder names are strings of up to 31 characters in length. They are not allowed to contain colons, but you should also avoid other special characters, as they can create problems and confusion. For example, if you include a slash in a folder name and then forget to double quote the name when you use it in a command, MacDOS will interpret the part beginning with the slash as a switch. Moreover, most MacDOS commands reject names beginning with a slash whether you double quote them or not. For similar reasons, backslashes should also be avoided. Also note that if you include a semicolon in a folder name, you will not be able to use it in the PATH system variable. In general, try to avoid characters which are used as separators.

A sequence of folders within folders separated by slashes is called a "path". If it starts with a backslash (ie. if it includes the root) the path is said to be "absolute", otherwise it is "relative". For instance, "\fold1\aDeeperFold" is an absolute path, while "aDeeperFold\still deeper" is relative.

Fortunately, in most cases you do not need to type the volume ID and the whole absolute path. You can direct MacDOS to memorise a particular absolute path and never have to specify it

again until you need to change it. In fact, you can tell MacDOS to memorise an absolute path for each mounted volume. Such a "default path" for a particular volume is called the "current folder" of that volume.

A backslash within a path directs MacDOS to "enter" a subfolder. That is, to go one level "down" (or "in"). In order to access one of the folders between the root and the current folder, you must also be able to tell MacDOS to go one level "up" (or "out"). For this purpose, MacDOS accepts the fictitious folder name "`..`": when you type a pair of periods (ie. dots) in place of a real folder name, MacDOS backtracks one item in the default path. Naturally, this means that you should not use "`..`" to name folders.

Let's bring everything together with some examples:

- You have two volumes: `1` and `2`.
- The current volume is `1`. That is, you are "attached" to volume `1`.
- The current folder of volume `1` is `"\aaa\bbb"` (the path which identifies a current folder is always absolute).
- The current folder of volume `2` is its root (ie. the default path is just "`\`").

Under these circumstances,

this	**is equivalent to**
`fold`	`1:\aaa\bbb\fold`
`2:fold`	`2:\fold`
`\fold`	`1:\fold`
`..\fold`	`1:\aaa\fold`
`..\bbb\..\bbb\..`	`1:\aaa`
`..\..`	`1:\`

Immediately after launching MacDOS, the default path is set to identify the folder which contains the MacDOS application file. Therefore, until you direct MacDOS to change the current volume or the current folder, your commands operate by default on the contents of the folder which contains MacDOS itself.

Creating Folders

You can create a new folder with the command `MKDIR` (which stands for MaKe DIRectory). You can also abbreviate `MKDIR` to `MD`.

`MD` accepts a single parameter which specifies the folder to be created. For example, you can create a new folder in the current folder simply by typing: `MD newFolderName`.

Removing Folders

You can remove a folder with the command `RMDIR` (ReMove DIRectory), which can be abbreviated to `RD`.

Similarly to `MD`, `RD` accepts a single parameter which specifies the folder. To remove a folder contained in the current folder, type: `RD oldFolderName`.

Note that you can only remove empty folders. The file `deldir.bat` in the original MacDOS floppy is an example of how you can remove non-empty folders.

Renaming Folders

You can change the name of a folder with the command `RENDIR` (REName DIRectory).

`RENDIR` accepts two parameters, to identify the folder to be renamed and to provide the new name.

Listing Folders

You can list the contents of a folder with two commands: `DIR` and `TREE`. They both accept a parameter which specifies the folder to be listed.

`DIR` provides the following information on each item (folder or file) in the folder:

- item name;
- size in bytes if the item is a normal file or volume ID of the target if it is an alias;
- file creator and type if the item is a file;
- date and time of last update.

For aliases, `DIR` displays a second line with information on the target. For chains of aliases, `DIR` displays information on the alias in the folder being listed and on the target, but not on the intermediate aliases.
`DIR` also displays the volume name and the absolute path of the folder being listed, as well as the total number of bytes occupied.

`TREE` displays the name of the items in a graphical way, so that the hierarchical structure of the folder is highlighted.

You can direct both `DIR` and `TREE` to select a subset of items by extending the parameter to include a wildcarded filename. In addition, several switches allow you to select what to list:

- files;
- folders;
- files of a certain creator;
- files of a certain type;
- hidden files;
- aliases.

Through the switch `/O`, you can also obtain lists in which the items are sorted on the basis of their name, size, last update, and whether they are folders or files.

Through the switch `/S`, you can display information on the whole hierarchy of folders and files contained in the specified folder.

If you "are lost" and would like to find out what the current folder is, type `CD`. You normally use this command to change the default folder (see below for a description), but without parameter it tells you where you are. Perhaps, the best way of always knowing "where you are" is to include the option `$p` in the `PROMPT`.

Changing the Current Folder

Immediately after launching MacDOS, the current folder is the folder which contains the MacDOS application file. This is true regardless of whether you launch MacDOS by double clicking on a text file with creator `'mDOS'`, on an alias of MacDOS, or on the application file itself.

You can change the current folder with the command `CHDIR` (CHange DIRectory), which can be abbreviated to `CD`. `CHDIR` accepts a single parameter which specifies the folder you want to make current. If the folder is on a different volume, MacDOS automatically attaches to the new volume before changing folder.

`CHDIR` without parameter tells you the current volume and folder.

Copying Folder Contents

You can copy all the items contained in a folder to another folder with the command XCOPY. The format of XCOPY is:

```
XCOPY sourceFolder destinationFolder
```

By default, XCOPY only copies the files and does not look inside subfolders. To copy everything, so that the destination folder is identical to the source both in hierarchical structure and contents, use the switch /E.

Other switches let you select files of a particular creator or type, or files updated after a certain date.

Note that XCOPY does not make a copy of the folder itself. Therefore, if you want to copy a folder with its contents, you must first create the destination folder. For example, to duplicate the folder aFold, type what follows:

```
mkdir "aFold copy"
xcopy/e aFold "aFold copy"
```

Special Folders

The Macintosh Finder gives you the impression that the desktop is the base of everything: volumes, folders, and files rest on the desktop, and when you move a file from a folder window to the desktop you have the impression that you are moving it "out". In reality, each volume (eg. a floppy) has its own desktop, and desktops are just folders.

Earlier on, you have learnt that each volume has a "root" folder which contains everything. If you wanted to give a name to the root folder, it would be reasonable to name it like the volume itself. Nevertheless, to type the root name would be a waste of energy, because it could only appear at the beginning of an absolute path. That is why in MacDOS (as in DOS) the name is dropped altogether, and the root folder is just identified by a backslash on its own.

The Macintosh Finder has implemented a similar strategy: when you double click on a volume icon (eg. the icon of a floppy), the Finder opens a window which has the same name as the volume and shows the contents of the root.

In each root there are two special folders named "Desktop Folder" and "Trash". With MacDOS you can see these two folders of the startup volume by typing:

```
cd 1:\          this attaches you to the root of the startup volume;
dir/a:h         this lists the contents of the root including hidden items.
```

The Finder tricks you into believing that there is a single desktop by displaying together the contents of the desktop folders of all mounted volumes. It does the same with the trash can. You do not see these two folders when you open a volume window (ie. when you open the root folder by double clicking on the volume icon) because the Finder hides them: they are there but you cannot see them.

If you insert a floppy and use DIR to list the contents of the root, you might find that one or both of these two special folders are missing. The reason is that they are not automatically created when the floppy is formatted. They are only created when they are needed the first time.

One final point needs to be made concerning the issue "one desktop per volume": dir "\Desktop Folder" only lists what is in the desktop folder of the current volume. Similarly, dir \Trash only lists what was trashed from the current volume. This way of displaying all desktop and trash folders separately might look like a limitation, but it gives you more control than the Finder. For example, it lets you selectively empty the Trash Can of a particular volume (you need a short batch program modelled on the example deldir.bat provided in your MacDOS disk). If you want to access all the desktop or trash folders together, you can always write a small batch program which scans all your volumes.

Section 6

Working with Files

6 Working with Files

This section describes the MacDOS commands which let you work with files.

Many MacDOS commands operate on files. To tell MacDOS where to look for them, you specify the folder which contains them. If you do not specify any folder, MacDOS looks for the files in the current folder.

Identifying a File

Like with folders, you identify a file through its name and its location. Therefore, to identify a file you must provide:

- The ID of the volume which contains the file.
- The name of all folders, one inside the other, which you need to open in order to "reach" from the root the file you are looking for.
- The name of the file.

To fully identify a file in MacDOS you need to type something like this:

```
1:\fold1\aDeeperFold\still deeper\myFile
```

and all the considerations made in the previous section concerning folders apply to files.

File names are strings of up to 31 characters in length. They are not allowed to contain colons, and you should also avoid other special characters, as they can create problems and confusion. For example, if you include a slash in a filename and then forget to double quote the name when you use it in a command, MacDOS will interpret the part beginning with the slash as a switch. Moreover, most MacDOS commands reject as bad switches names beginning with a slash, whether you double quote them or not. For similar reasons, you should also avoid backslashes. Also note that if you include a comma in a filename, you will have problems in copying it, because the command `COPY` accepts comma-separated filenames. In general, try to avoid characters which are used as separators.

Creating New Files

There are three ways of creating a new file with MacDOS:

- Create a file with the command `OPEN` and then write into it lines of text with the command `WRITE` (please refer to one of the following sections for the details).
- Use the command `LOG` to record all the commands executed by MacDOS (please refer to the "Getting Info" section of this User's Guide).
- Redirect the output of a MacDOS command to a disk file (please refer to "The Environment" section of this User's Guide).

By default, the files created are of type `'TEXT'` and creator `'ttxt'` (ie. TeachText), but you can change the creator by updating the system variable `CREATOR`.

Removing Files

You can remove files with the commands `DEL` and `ERASE`. The two commands have different names but are identical for what concerns functionality. Therefore, what is said about `DEL` always applies 100% to `ERASE`.

`DEL` accepts a single parameter which specifies the file you want to delete or the folder you want to empty of files. If you include wildcard characters in the parameter, `DEL` deletes all the files

whose names match the parameter. You also have the possibility of selecting files on the basis of their creator, type, or both.

To delete all the files in a particular folder, you can pass to DEL a "catch all" filename or simply the name of the folder:

```
del D:\theFolder\*
del D:\theFolder
```

In either case, MacDOS prompts you for confirmation.

Note that DEL permanently removes the files from the system rather than simply moving them to the Trash. To limit the danger of such a drastic behaviour, MacDOS prompts you for confirmation before deleting each individual file. Nevertheless, you can switch this safeguard off by executing the command CONFIRM OFF . To re-enable the prompting for a single DEL command, add the switch /P. To re-enable it permanently, execute the command CONFIRM ON .

Listing Files

You can list files with two commands: MORE and TYPE.

MORE displays the content of the data fork of files of type 'TEXT' one page at a time. At the end of each page, MORE waits for you to type any character before continuing. You can also interrupt the listing by pressing cmd-dot (or cntl-C). MORE is a special command, in that it expects that you specify a file as input redirection rather than as a parameter. So, if you want to list the file 2:\aFold\theFile, you must type:

```
more <2:\aFold\theFile
```

rather than simply:

```
more 2:\aFold\theFile
```

TYPE, when used in its basic form, also lists the data fork of files of type 'TEXT' on the monitor screen. The differences are that TYPE expects a file specification as a parameter and only stops when it reaches the End Of File (EOF). You can pause and resume long listings with cntl-S and cntl-Q respectively.

To list the data or resource fork of any file (not only text files), use TYPE with the switches /H and /R respectively. TYPE then lists the requested fork in HEX, as well as in ASCII, 16 bytes per line.

Copying Files

To copy files, you can use the commands COPY and XCOPY. This section only describes COPY. Please refer to the section "Working with Folders" for a description of XCOPY.

In its simplest form, COPY makes a duplicate of a file which already exists and places it in the current folder: copy aFileSpec . The difference with the "Duplicate" menu item provided by the Finder is that COPY's duplicate has the same name as the original file. Therefore, if you COPY a file which already is in the current folder, your command does not have any visible effect and you do not get any duplicate at all.

If you pass to COPY a second file specification, you can:

- specify a different name for the duplicate;
- place the duplicate anywhere in the system rather than in the current folder.

The two file specifications are usually called "source" and "destination". With this format, you can reproduce the Finder's Duplicate function as follows:

```
copy fileName "fileName copy"
```

A duplicate of the file `fileName` in the current folder (ie. the front window) is created and named `fileName copy`. Note that the double quotes are only necessary because the new name contains a space.

If you use the destination to specify a folder rather than a file, the duplicate file is created with the same name as the original. In other words, the two following commands are equivalent:

```
copy fileName destFolder\fileName
copy fileName destFolder
```

Still using the second parameter to specify a folder, you have three ways of `COPY`ing several files at a time:

- By replacing the single source with a series of file specifications separated by commata, you `COPY` all the individual source files to the destination folder:
  `copy file1,aFold\file2,2:file3 destFolder`

- By wildcarding the source, you `COPY` all the files with matching filenames to the destination folder:
  `copy sourceFolder\p*a??? destFolder`

- By using the first parameter to specify a folder rather than a file, you `COPY` all the files in the source folder to the destination folder:
  `copy sourceFolder destFolder`

If you specify more than one file as a source parameter but use the second parameter to specify a single file, `COPY` merges all the sources to produce the destination. This can be useful to append to each other several data or text files.

`COPY` supports several options:

- Append the source to an existing destination.
- Only copy files of a given Creator or Type.
- Only copy the Data or Resource fork of files.
- Display a Prompt before overwriting destination files.
- Only copy files which either do not exist in the destination folder or do exist but have an older Update date and time.
- Verify that the copies are correct.

Renaming Files

To rename files, you can use the commands `REN` and `RENAME`. Apart from their name, the two commands are identical.

In its simplest format, `REN` works as follows:

```
ren fileSpec newName
```

The first parameter specifies the file to be renamed and the second parameter tells MacDOS what the new name should be.

If the first parameter ends with a wildcarded filename, all the files with matching names are renamed. In that case, the second parameter is used to replace in each name the part which precedes the first wildcard character. For example:

```
ren abc* xyz
```

renames `abc` to `xyz`, `abcf` to `xyzf`, `"abc whatever"` to `"xyz whatever"`, etc. In this form, `REN` can also be used to insert a string before all filenames:

```
ren * prependThis
```

MacDOS also accepts wildcards in the second parameter. In that case, MacDOS matches the groups of wildcards and replaces the corresponding parts.

If the first parameter specifies a folder rather than a file, MacDOS renames all the files in the folder. Therefore, the two following commands have identical results:

```
ren aFold\* x*z
ren aFold x*z
```

In both cases, MacDOS prepends an 'x' and appends a 'z' to the name of each file in the folder aFold.

Through a series of options, REN also lets you select files on the basis of their creator and type. You can also specify whether the name selection should be case sensitive or not.

Changing File Attributes

You can use the command REN and (RENAME, which is identical) to change more than just the names of files. In particular, you can:

- set new creators;
- set new file-types;
- set the date and time of the last update to the current date and time;
- hide a visible file and viceversa.

This functionality is controlled through switches. For example, "ren zz /h" hides the file named zz, and "ren/c!????" changes the creator of all files in the current folder to '????' (be careful!).

Printing Text Files

You can print files of type 'TEXT' with the command PRINT. The format of PRINT is: print fileSpec.

The file specification can be wildcarded or can be replaced with a folder specification, in which case all text files in the folder are printed. Without parameter, PRINT prints what is displayed in the console window.

Before PRINTing, you can specify the page format with the standard "Page Setup ..." item of the File menu.

Instead of displaying the standard Print dialog every time you print, MacDOS prints with a default selection, so that you can PRINT several files without having to click for each one of them. With the /D switch you can direct MacDOS to display the standard "Print..." dialog before printing the first file.

Direct File I/O

MacDOS lets you OPEN, READ or WRITE, and CLOSE files of type 'TEXT'.

You can OPEN a file for reading, writing, or appending. In each case, one line of text at a time is transferred in each I/O operation, either from disk to memory or from memory to disk. When READing, you can store the line into a global variable for later processing, or display it for immediate view.

After you have READ from a file the last line, MacDOS automatically closes it.

By default, MacDOS creates new files with creator 'ttxt' (ie. TeachText), but you can change it by setting the system variable CREATOR to your preferred creator.

Type OPEN without parameters to display the list of files which are currently open.

Section 7

Global Variables

7 Global Variables

Global variables are just a way of saving character strings for later use. They are referred to as "global" because they are accessible from within all batch programs as well as interactively.

MacDOS automatically generates and uses some variables called "System Variables".

Creating and Setting Variables

You can set global variables interactively or in batch programs with the command `SET`. When you `SET` a variable for the first time, MacDOS automatically creates it. The format of `SET` is very simple:

```
set variable name=whatever string you like
```

Any string can be used to name such user-defined variables, but be aware that:

- If you include a percent sign, this will confuse MacDOS when you try to use the variable in a command, because percent signs are used to access the value of variables (see below).
- If you include double quotes or other special characters like colons, slashes, and backslashes you might run into problems later when you use the variable in batch programs. At the very least, you will find it confusing when reading your program.
- When recording a variable name, MacDOS ignores leading and trailing spaces and tabs. That is, you will not be able to create a variable name which begins or ends with blanks.
- MacDOS also simplifies all sequences of spaces and tabs found within variable names, so that words forming a variable name are always separated by single spaces.

The string which follows the equal sign is stored as it is, including leading and trailing blanks.

When you `SET` an existing variable, MacDOS simply replaces the current value with the new one.

Displaying and Removing Variables

To display a list of the existing variables and their content, just type the command `SET` without parameters. MacDOS then displays all the variables in the order in which they were created.

You can also display the string stored in a single variable by typing:

```
ECHO %varName%
```

When you do not need a user-defined variable anymore, it is a good practice to remove it from the system. To do so, `SET` the variable to an empty string by typing return immediately after the equal sign. System variables are special, in that they cannot be removed from the system. When you `SET` a system variable to the empty string, MacDOS does not remove it but sets it back to its default value.

Retrieving Variables

You can access the string stored in a variable by inserting in a command the variable name enclosed between a pair of percent signs. For example, you can define the variable `HOME` by executing `SET HOME=%WHERE%` and then attach back to that folder with `CHDIR %HOME%`. Similarly, if the variable `RET_LBL` contains a label name, you can jump to that label with the command `GOTO %RET_LBL%`.

If you type two consecutive percent signs, MacDOS recognises that you are not attempting to de-reference a variable or a replaceable parameter and handles the pair of characters as a single percent sign.

Operations on Variables

MacDOS provides a series of commands which let you manipulate global variables. They are: INCR, DECR, SSTR, and TOUPPER. If you need to keep the original value, use SET to save it before executing any of these commands.

INCR

INCR has the purpose of incrementing numeric values, adding numbers, and extending or concatenating alphanumeric strings.

```
incr numVar
incr numVar by 7
incr numVar by -3
```

are examples in which INCR is used to increment the numeric value stored in the global variable numVar. In the last example, the final value is actually less than the initial one, because the increment is negative.

```
incr numVar by %anotherNumVar%
```

shows how INCR can be used to add anotherNumVar to numVar.

```
incr var by " append this"
incr -var by "prepend this "
incr var by anAlphaVar
incr alphaVar by 3
incr -alphaVar by 3
```

are examples of how INCR can be used to attach strings to each other. In the last two examples, three spaces are appended and prepended to alphaVar respectively.

DECR

DECR has the purpose of decrementing numeric values, subtracting numbers, and truncating or subtracting alphanumeric strings.

```
decr numVar
decr numVar by 7
decr numVar by -3
```

are examples in which DECR is used to decrement the numeric value stored in the global variable numVar. In the last example, the final value is actually greater than the initial one, because the decrement is negative.

```
decr numVar by %anotherNumVar%
```

shows how DECR can be used to subtract anotherNumVar from numVar.

```
decr var by " remove this from the end"
decr -var by "remove this from the beginning"
decr var by anAlphaVar
decr alphaVar by 3
decr -alphaVar by 3
```

are examples of how DECR can be used to subtract strings from each other. In the last two examples, the three last and first characters are removed from alphaVar respectively.

SSTR

SSTR extracts a substring from a variable (the name SSTR stands for SubSTRing). In order to identify what should remain and what should be removed, SSTR uses a string as delimiter. Its format is:

```
sstr var delim
```

where delim is the string which SSTR uses to split the variable.

By default, SSTR searches for the delimiter from the beginning of the variable string, and extracts the substring on the left of the delimiter. Optionally, you can direct SSTR to search from the end rather than from the beginning, and to keep the substring on the right of the delimiter rather than on its left.

TOUPPER

TOUPPER converts the value of a variable to upper case. It is useful when you want to compare strings and filenames ignoring upper and lower cases. Note that diacritical marks and international characters are converted correctly: ü to Ü, æ to Æ, ø to Ø, etc.

System and Special Variables

Currently, MacDOS implements seven system variables: PROMPT, DIRCMD, PATH, WHERE, CREATOR, QUOTE, and FONTSIZE. MacDOS also uses the variables DOSERR, SHOWAE, and TIMEOUT, which behave like user-defined variables and can therefore be removed.

PROMPT

stores the format of the string displayed by MacDOS to prompt you for a new command.

By using a dollar sign as an escape character, PROMPT can memorise requests to display special information, like the current date and time, the specification of the current folder, the current volume ID, and the MacDOS version.

Instead of setting the variable PROMPT with the command SET, you should use the command PROMPT. This would ensure that you set the prompt string to a valid value.

By default, the prompt string consists of the current volume ID followed by a "greater than" sign (PROMPT=ng). Please refer to the command reference section on PROMPT for a list of dollar-options.

DIRCMD

stores the default options for the commands DIR and TREE.

By default, DIR and TREE list all visible files and folders in extended format, non recursively, non sorted, and not case sensitive. Please refer to the command reference section on DIR for a list of valid options.

PATH

stores the list of folders that MacDOS searches when looking for batch programs, applications, AppleScripts, MacDOS extensions, and abbreviation files.

The folder specifications are separated by semicolons.

When you direct MacDOS to execute a program or look for an abbreviation file, MacDOS searches the current folder for a file of the appropriate type with the requested name. It then searches the folders specified in PATH, scanning the list from left to right.

MacDOS initialises the system variable PATH to the empty string. Therefore, by default MacDOS only searches the current folder.

To set the variable PATH you can also use the command with the same name. To add a folder specification to the existing list, you can execute any of the following commands:

```
set path=%path%;newFolderSpec
path %path%;newFolderSpec
```

```
incr path by ;newFolderSpec
```

WHERE

always contains the specification of the current folder.

MacDOS updates `WHERE` every time you change volume or directory.

This variable lets you "go back" to the initial folder after using `CHDIR` to attach to other folders and volumes. To do this, you must save `WHERE` by copying it to another variable before attaching to the new folder. For example:

```
set back=%where%
chdir anotherFolder
...
chdir %back%
```

CREATOR

stores the file creator that MacDOS uses when creating new files. By default, the creator is `'ttxt'` (ie. TeachText).

You can set it to the creator of your preferred editor or word processor. You can also set it to `'mDOS'` if you want to create batch programs. The alternative would be to create the files and then use `REN` to change their creators.

Note that MacDOS only uses the first four characters of this variable. Therefore, you can use the rest of the line for comments.

QUOTE

stores the character used by MacDOS to delimit file and folder names which contain spaces and other special characters. The default is the double quote character.

By changing `QUOTE`, you can access file names which contain double quote characters:

```
set quote=#
rename #with "quotes"# #without quotes#
del bla"bla
set quote=
```

FONTSIZE

stores the point size of the characters used in the console window.

The default is 9, but the values 12, 18, 24, and 36 are also legal. Other values are rounded down to one of the permitted sizes, except values lower than 9 which are rounded up.

DOSERR

stores the code of the last error. As error codes are always positive numbers (see appendix A), zero can be used as a no-error condition.

MacDOS updates this variable whenever an error (not a warning) occurs, but does not reset it when a command completes successfully. Therefore, it is your responsibility to remove it or initialise it to zero.

You can obtain the error message corresponding to a particular error code with the command `SHOW`.

SHOWAE

tells MacDOS whether it should display incoming Apple Events.

MacDOS only checks whether this variable exists or not, completely ignoring its value. For a list of Apple Events supported by MacDOS, please refer to the section "MacDOS Scripting".

TIMEOUT

Stores the number of seconds that MacDOS waits for piped messages to go through a chain of MacDOS extensions. Please refer to the section "MacDOS Extensions" for the details.

Section 8

Batches and Programs

8 Batches and Programs

MacDOS recognises four types of "executable" files:
- Applications: Files of type `'APPL'`.
- Compiled AppleScripts: Files of type `'osas'`.
- Text AppleScripts, of type `'TEXT'` and creator `'ToyS'`.
- Batch programs, of type `'TEXT'` containing MacDOS commands to be interpreted and executed one line at a time.

You have two ways of telling MacDOS what program to execute:
- by writing its file specification at the beginning of a command line;
- by executing the command `call fileSpec`.

The two strategies are only different if you are already executing in batch: in the first case, control is simply transferred to the new program; with `CALL`, execution of the current batch resumes after the application has been launched or the new batch program has terminated.

Instead of having to type full file specifications, you can take advantage of the system variable `PATH` and just type the name of the application or batch program. If you keep all applications and batch programs (or their aliases) in a small number of folders, you will never have to type a full file specification at all.

Note that MacDOS, unlike DOS, does not automatically add the extension `.EXE` when looking for executable files. Neverthess, similarly to DOS, MacDOS automatically adds the extension `.BAT` when looking for batch programs.

Introduction to Batch Programs

After opening a batch file for reading, MacDOS reads and attempts to execute one line of text at a time. In its simplest form, a batch program is therefore just a way of storing a series of MacDOS commands so that you do not need to type them over and over again.

Nevertheless, there are a series of features which make batch programs very useful:
- Several MacDOS commands are only available within batch programs.
- You can control the execution of batch programs through launch parameters.
- You can setup a batch program in such a way that it behaves like a Macintosh application.
- You can document your algorithms and the reasoning behind them by adding lines of comments.

To add lines of comments, you can use the command `REM` (for REMark). `REM` tells MacDOS to ignore the rest of the line and continue with the next one. You can also use an exclamation mark ('`!`') in the same way.

If you redirect the output of a batch program, MacDOS automatically applies the redirection to each individual command in the batch file, including nested batches. Note that the output of batch programs cannot be piped.

Jumps and Labels

Normally, MacDOS executes the lines of a batch program in sequence, as they appear in the file. You can direct MacDOS to interrupt the sequence and resume execution somewhere else in the file by using the command `GOTO label`.

Almost any word or number can be used as a label but, to be recognised and accepted as such, it must satisfy the following conditions:

- It cannot contain spaces or tabs.
- It cannot begin with a percent sign.
- It must be the first word in a line which begins with a colon.
- It must be unique within the file.

Before executing a batch program, MacDOS scans the whole file looking for lines which begin with a colon. For each one of such lines, MacDOS builds a "label entry" containing the first word after the colon and the file offset of the next line. Note that MacDOS ignores whatever follows the label, so that you can use the rest of the line for commenting. Also note that the colon identifies the label but is not part of it.

When MacDOS encounters a `GOTO` during batch execution, it scans the label entries looking for the requested label. If it finds an entry with the matching label, it then resumes execution of the batch program from the line with the file offset found in the entry.

Replaceable Parameters

Whether you just type a batch filename or use the command `CALL`, you can pass to the batch program a series of parameters.

The parameter `%0` stores the name of the batch file itself, while the parameters `%1` to `%9` are available for you to define as you need.

Each one of these replaceable parameters consists of a string and is accessible from within the batch program through a number preceded by a percent sign. For example, if you type:

```
myBatch name "a full address"
```

the program `myBatch` will be able to access the string `name` as `%1` and the string `a full address` as `%2`. The string `myBatch` will be available as `%0`.

Despite the limitation in the number of parameters, MacDOS lets you pass to a batch program up to 24 strings. You can access the additional 15 strings by using the command `SHIFT`. `SHIFT` copies `%1` to `%0`, `%2` to `%1`, ..., and `%9` to `%8`; it then stores the 10th string into `%9`. By "pushing in" one new string at a time in this way, you can acess all 15 additional strings.

Conditional Statements

MacDOS lets you implement conditional statements with the command `IF`. The general format of `IF` is **`if`** `condition` **`do`** `aCommand`, where `aCommand` is executed only if `condition` is true.

Valid conditions are:

```
string1 == string2
exist fileSpec
existdir folderSpec
not existdir folderSpec
```

As you can see from the last example, you can also invert a condition by inserting a **`'not'`** between the `IF` and the condition.

Loops

Very often, a particular sequence of commands needs to be repeated for each file belonging to a particular set or until certain special conditions are verified. To serve this purpose, MacDOS provides the command: `FOR`.

FOR lets you define a set of files and then operate on each one of them. In its simplest form, FOR executes a single command included in the same line:

```
for %name in (set) do aCommand
```

where set is a comma-separated list of file specifications, and aCommand can refer to the current file through the special variable %name.

To execute several commands for each file, you can use an extended format of FOR:

```
for %name in (set) do begin
    ... several commands which can refer to the current file through the
        global variable %name% ...
    next name
```

The most general way of implementing loops is with a combination of an IF and a GOTO. For example, if you need to repeat certain operations until a certain condition is verified, you can do as follows (equivalent a WHILE or a FOR in C or a WHILE-DO in Pascal):

```
set k = 1
:LOOP_LBL
    if k == 5 goto BREAK_LBL
    ... here you do what you need to do ...
    incr k
    goto LOOP_LBL
:BREAK_LBL
```

Alternatively, you can place the IF at the end of the loop (equivalent to a DO in C or Fortran or a REPEAT-UNTIL in Pascal):

```
set control=initial
:LOOP_LBL
    ... here you do what you need to do ...
    ... and also set the control variable in order to exit the loop ...
    if control==done goto BREAK_LBL
    goto LOOP_LBL
:BREAK_LBL
```

Error Handling

Beside reporting error messages in clear, MacDOS also stores error codes into the global variable doserr. To convert the error codes to their corresponding messages, you can use the command SHOW.

The command ONERROR lets you trap all the errors which occur while executing a batch program. This is particularly useful when you execute a batch program with ECHO OFF, because in that case MacDOS suppresses error and warning messages.

ONERROR lets you specify the label of an error handling procedure. For example,

```
onerror label
... first command for which you want to trap errors ...
... second command for which you want to trap errors ...
...
onerror
```

is functionally equivalent to the following less readable construct:

```
set doserr=0
... first command for which you want to trap errors ...
if not %doserr% == 0 goto label
... second command for which you want to trap errors ...
if not %doserr% == 0 goto label
...
set doserr=0
```

User Interface

While executing in batch, MacDOS normally displays all commands as they are executed. With the command `ECHO OFF`, you can direct MacDOS to suppress both the echoing of the commands and their output, including warning and error messages. You can then re-enable the display with `ECHO ON`.

If you only want to suppress the output of a single command, you can do so by attaching an '`@`' to the front of the command name.

By typing as first line of a batch program the command `@echo off`, you make the program completely silent.

In order to display variable values and other messages while echoing is disabled, you can use `ECHO` as illustrated in the following examples:

```
echo Now processing file No %fileNum%
echo File %1 found: it is in folder %foldSpec%
```

After `ECHO`ing an intermediate result, you might need to suspend execution of a batch program and wait for confirmation before continuing. For this purpose you can use the command `PAUSE`. When MacDOS encounters `PAUSE`, it displays the message:

```
press any key to continue ...
```

and waits for your reply. If you type cmd-dot or cntl-C, MacDOS then asks

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

You can then terminate the job by replying '`Y`' or '`y`'.

Batches that behave like Applications

Any file of type `'TEXT'` is accepted by MacDOS as a valid batch program. Nevertheless, if you want to automatically launch MacDOS by double clicking on the icon of a batch file, you need to set its creator to `'mDOS'`.

By using the command `EXIT` as last line, you can also automatically terminate MacDOS and return to the Finder at the end of the batch program.

Here is a simple program that is actually used every day to back up this User's Guide while it is being written:

```
confirm off
copy "\MacDOS folder\product documents\manual" 2: /u/v
exit
```

It copies to a floppy all files which have been updated since the last backup.

You can preset to `'mDOS'` the creator of text files by setting the system variable `CREATOR`:

```
set creator=mDOS
```

Alternatively, you can change to `'mDOS'` the creator of existing files:

```
ren/c!mDOS fileSpec
```

Note that MacDOS executes the batch program `autoexec.bat` (see below) before executing the batch program that you have double clicked.

If you keep MacDOS running all the times, instead of aliasing the MacDOS application into the folder "Startup Items", you might like to replace all the items in that folder with a small batch program which `CALL`s them all. They would be launched in the order in which you call them (MacDOS will always be the first one, though).

Section 9

Customising MacDOS

9 Customising MacDOS

autoexec.bat

MacDOS launches the batch program named `autoexec.bat` every time it starts. Therefore, `autoexec.bat` is effectively the MacDOS' preference file. In it you can:

- Set your preferred date format with `DATE`.
- Set a 12 or 24 hour clock with `TIME`.
- Make your life a bit more dangerous with `CONFIRM OFF`.
- Make file copies faster with `VERIFY OFF`.
- Ensure that you do not overlook any error messages by forcing MacDOS to display an alert box whenever an error or warning condition occurs. For this purpose use the command `ALARM ON`.
- Use `PATH` to tell MacDOS in which folders you keep your executable files (applications and batch programs).
- Set your preferred command prompt with `PROMPT`.
- `SET` the system variable `DIRCMD` to your preferred defaults for listing folders.
- Assign your literal volume IDs with `SUBSTVOL`.
- Use `CALL` to execute additional initialisation batch programs and to launch other Mac applications.
- Use `TYPE` to display a text file containing reminders of things to do.
- Store the MacDOS directory name into the global variable `HOME` by `SET`ting it to `%WHERE%`.
- Attach to your working folder with `CHDIR`.
- `ECHO` to the screen a welcoming message.
- Set up your abbreviations by `SET`ting the variable `ABBREV`.
- Change the size of the characters displayed in the console window by `SET`ting the system variable `FONTSIZE`.
- Change the character used for quoting long filenames by `SET`ting the system variable `QUOTE`.
- Change the file creator used for output text files by `SET`ting the system variable `CREATOR`.
- Decide to be notified of all incoming Apple Events by `SET`ting the variable `SHOWAE`.

Date

You can change the system date with the command `DATE`. With the same command you can also choose the format for entering and displaying dates.

For example, you can choose to use an American format for the dates by typing:

```
date mm/dd/yy
```

If you then change the system date with the command

```
date 5/23/94
```

MacDOS will display the new date as:

```
05/23/94
```

Alternatively, you can compose your own date format:

```
date yyyy;mmmm,dddd
```

If you then change the system date with the command

```
date 94;5,23
```

MacDOS will display the new date as:

```
1994;0005,0023
```

Time

When you change the system time with the command `TIME`, you can at the same time choose between a 12 and a 24 hour clock format. You can also choose the separators used between the time fields.

For example, if you set the system time with the command

```
time 17.20.10
```

MacDOS will display times with a 24 hour clock and dots as separators.

If you set the system time to midnight, MacDOS interprets the command as a request to change the clock format. This means that you cannot actually change the system time to midnight (you will have to wait at least one second !), but gives you the possibility of choosing the clock format without altering the current system time.

Therefore,

```
time 00:00:00a
```

leaves the current time unchanged and switches to a 12 hour clock. Similarly,

```
time 24:00:00
```

switches to a 24 hour clock without touching the system time.

Alerts

By default, MacDOS reports errors and warnings with a line of text. If you execute the command `ALARM ON`, MacDOS replaces the line of text with an alert box and waits for you to click on its OK button before continuing.

`ALARM OFF` reverts the error and warning reporting to the default.

With `ALARM ON`, you can execute complex batch programs without having to monitor their progress, because you know that MacDOS automatically suspends execution if it encounters a problem.

Confirmation

When you use the Finder to move files between folders, the Finder always asks for confirmation before overwriting existing files. Also when you empty the Trash you have to confirm it.

By default, MacDOS does the same and prompts you for confirmation. This is particularly important when `DEL`eting files, because MacDOS removes them from the system instead of copying them to the Trash.

You can direct MacDOS not to ask for confirmation by executing `CONFIRM OFF`.

You can then revert to the safer default with the command `CONFIRM ON`.

It is a good and safe practice to disable confirmations only in special cases and for short periods of time. Be aware of the fact that if you disable confirmation within a batch program, it remains disabled after the batch has completed execution. It is therefore advisable to re-enable safeguards within the same batch file that disables them.

Verification of File Copying

By default, MacDOS verifies the correctness of all `COPY` and `XCOPY` operations. You can disable verification by executing `VERIFY OFF`. You can then revert back to the safer default with `VERIFY ON`.

With `VERIFY ON`, MacDOS reads the copy and compares it with the original. Naturally, this slows down the whole operation. Nevertheless, it is advisable to put up with the little delay, especially when copying to floppies: only when the files are read back correctly you can be sure that they were written correctly.

Prompt

The command `PROMPT` tells MacDOS how to prompt you for a new command. Through `PROMPT`, you can request MacDOS to display special information like the current date and time, the specification of the current folder, the current volume ID, and the MacDOS version.

`PROMPT` is in fact a command which is totally equivalent to the `SET`ting of the system variable `PROMPT`. The only advantage of using the command instead of a straight `SET` is that the command performs a syntax check on the prompt string.

Literal Volume IDs

Instead of using straight volume reference numbers to identify volumes, you can use the command `SUBSTVOL` to define literal IDs. To a large extent, this is just a cosmetic change. Nevertheless, it can be of great help if you want to port batch programs originally developed for DOS.

Abbreviations

MacDOS supports an abbreviation mechanism that lets you type short words to insert long strings into the command line.

To use this mechanism, you must:
- Store your abbreviations into a file.
- Tell MacDOS how to find the file.
- Tell MacDOS what abbreviation you want to have expanded.

Preparing an Abbreviation File

With any text editor or word processor, create a file of type `'TEXT'` and insert lines like:

```
abbr this
```

where `abbr` is the abbreviation and `this` is the string to be abbreviated.

The rules governing the formatting are:
- `abbr` must be at the beginning of a line.
- `this` is delimited on the left by the first space or tab (ie. the first blank) which follows `abbr`, and on the right by a Carriage Return. Therefore, additional spaces or tabs between `abbr` and `this` will be considered to be part of `this`. Moreover, there can only be one abbreviation per line.
- Only the first 199 characters of each line are interpreted. The rest of the line is ignored (ie. the string to be abbreviated is truncated).
- Lines beginning with blanks or containing no blanks at all are ignored and can be freely used for commenting.

Additionally, be also aware that:
- Abbreviations are case sensitive (eg. 'Abc' is different from 'abc' and 'aBc').
- Redefinitions are ignored. Therefore, you will only be able to use the first definition with a particular `abbr`.
- The size of abbreviation files is limited to approximately 32kByte after removing the comments (but before removing badly formatted or duplicate abbreviations).

Telling MacDOS where the Abbreviation File is

Set the global variable ABBREV to the full path which identifies the abbreviation file (without quoting it, because SET accepts all characters after the equal sign). If the abbreviation file is in a folder included in the PATH variable, you only need to set ABBREV to its filename.

When MacDOS sees an assignment to ABBREV, it reads the file, stores the valid abbreviations in a block of dynamic memory, sorts them, and removes the duplicates.

Expanding the Abbreviations

To expand an abbreviation, type the abbr at the beginning of the command line or following a space. Then press shift-tab. MacDOS then replaces abbr with the corresponding this. If MacDOS does not find any matching abbr, it just ignores your request.

If you type only some initial characters of abbr, MacDOS can still perform the replacement as long as it finds a single match in the abbreviation file.

If MacDOS finds more than one matching abbreviation, it lists them all and re-prompts you. MacDOS also automatically extends your partial abbreviation if possible.

Example

In the file "1:\aFold\abbrev file" you stored the following lines:

```
rain   Rainbow Hill
 This line begins with a space. Therefore, it is a comment.
 The following line is empty. Still commenting...

aaaa Silly abbreviation
aaBc Ditto
```

To activate the abbreviations, type the command:

```
prompt>set abbrev=1:\aFold\abbrev file
```

To see the active abbreviations, type a shift-tab immediately after the MacDOS prompt. You will see:

```
  aaBc    "Ditto"
  aaaa    "Silly abbreviation"
  rain    "  Rainbow Hill"
```

If you now type (either at the beginning of the command line or after a space) aaBc followed by a shift-tab, MacDOS will replace it with the string Ditto. The same will happen if you only type aaB. If you type aa, MacDOS will display the two lines with the abbreviations aaaa and aaBc.

Finally, if you type a single 'a' before typing shift-tab, MacDOS will extend it to 'aa' before typing the list of matching abbreviations. That is, MacDOS will recognise that all the abbreviations that match 'a' also match 'aa' and will therefore append the second 'a' to your partial abbreviation.

Safe Hacks

This section explains a couple of simple changes that you can safely make to MacDOS with ResEdit.

Starting Up

You can remove the initial splash screen (sigh!) by deleting or renumbering the 'PICT' resource number 129 (or 130 for black & white).

Shutting Down

When you execute the commands `RESTART` and `SHUTDOWN`, MacDOS displays a little picture with a farewell message. You can replace the corresponding `'PICT'` resource number 131 with a picture of your liking.

More Help

You can add help messages by inserting resources of type `'TEXT'`. When you type `"help subject"`, MacDOS looks for a `'TEXT'` resource named `subject`. Therefore, you only need to add the necessary text resource and MacDOS will pick it up.

When you type `"help ?"`, MacDOS lists the names of all resources of type `'TEXT'` beginning with 128 and stopping when it finds the first missing number. Therefore, if you want to have the help items in alphabetical order, you will have to insert your resource in the appropriate place by "shifting down" the following ones.

If you want to get help on any file, you only need to add to that file a resource of type `'TEXT'` named "help". You will then be able to type `"help file name"`. It will work irrespective of the file type. This is possible because MacDOS, before returning the error message "No help available", looks for a file with the requested name and checks whether it contains a `'TEXT'` resource named "help".

Changing Fonts

When displaying characters in 9 and 12 points, MacDOS uses a modified version of Monaco with redefined zeroes and capital 'i's. The bitmapped fonts are stored in two resources of type `'FONT'`. You can replace them with different fonts, but MacDOS will only maintain proper formatting if they are monospaced.

Section 10

MacDOS Scripting

10 MacDOS Scripting

MacDOS is AppleScript-able. If you use Apple's standard Script Editor to access MacDOS' dictionary, you will obtain the following:

Required Suite:
Events that every application should support

open: Open the specified object(s)
open alias -- *list of objects to open*

print: Print the specified object(s)
print alias -- *list of objects to print*

quit: Quit application
quit

run: Sent to an application when it is double-clicked
run

MacDOS Suite:

dosExecute: Directs MacDOS to execute a command
dosExecute string -- *the command to be executed*

setVar: Sets a MacDOS global variable
setVar string -- *name of the variable*
to string -- *value to be assigned*

getVar: Obtains the value of a MacDOS global variable
getVar string -- *name of the variable*
Result: string -- *the value of the variable*

open executes the specified file[s] of type `'TEXT'` as batch program[s].

print prints the specified file[s] of type `'TEXT'`.

quit quits MacDOS.

run launches MacDOS.

dosExecute parses and executes the parameter as if it were a command entered from the keyboard. In particular, dosExecute followed by the name of a text file directs MacDOS to execute that file as a batch program.

setVar sets a MacDOS global variable to a given string.

getVar returns the value of a MacDOS global variable. This is particularly useful to access the variable `DOSERR`, which contains the error codes generated by MacDOS commands.

MacDOS only asks for input from the keyboard if the client application allows it (this is always the case if you use Apple's Script Editor to write your scripts).

If the Apple Event disallows interaction with the user, MacDOS follows the following rules:

- The input normally expected from the keyboard is handled as if the user had typed a `CNTL-C`. This causes the current command to be aborted as soon as an input is required.
- Error reports are always sent to the console window as strings of text regardless of whether `ALARM` is `ON` or `OFF`.
- The `/D` switch of the `PRINT` command is ignored so that the standard Print dialog is never displayed. Also when requested to print the console window, MacDOS does not display the standard Print dialog.

By setting the global variable `SHOWAE` to any value, you can direct MacDOS to display the class and eventID of all Apple Events received (actually, all High Level Events). This lets you check that the originating application is working correctly.

For debugging purposes, MacDOS displays in the console window the name of all batch files started via Apple Events.

Section 11

MacDOS Extensions

11 MacDOS Extensions

You can extend the functionality of MacDOS by creating your own "pipable" filter applications. Such applications accept as input the output of MacDOS commands and send their results back to MacDOS for display in the console window.

To work as a filter, an application must:
- Be of type `'APPL'` and creator `'mFLR'`.
- Be able to run in the background.
- Support the MacDOS communication protocol to exchange messages with other processes.

This section only describes how to use filters that already exist. If you are interested in creating your own new filters, please refer to Appendix B.

Basically, there are two ways in which you can use filters within the command line:

```
filter1 < inFile
```

or

```
command | filter1
```

In the first case, a text file is fed into filter1, while in the second case, the output of a MacDOS command is sent to filter1. In both examples, the output of filter1 is sent to the console window.

Commands and filters have in general one or more switches. Also note that the position of file redirection within the command line is irrelevant: output redirection automatically applies to the last (ie. rightmost) filter and input redirection to the first one.

Some examples of valid chains of commands and filters are:

```
filter1 <inFile >outFile | filter2
```
Data flow: inFile -> filter1 -> filter2 -> outFile

```
filter1 | filter2 <inFile
```
Data flow: inFile -> filter1 -> filter2 -> console

```
dir /b/a-d | myFilter mode1 >outFile
```
Data flow: MacDOS -> myFilter -> outFile.
`DIR` sends its output to myFilter and myFilter executes with the "mode1" option.

```
\filterDir\filter1 <inFile | more
```
Data flow: inFile -> filter1 -> console
MacDOS pages the output of filter1 to the console window

```
help filter1 | aSubDir\filter2 | filter3
```
Data flow: MacDOS -> filter2 -> filter3 -> console
MacDOS sends to filter2 the help information provided by filter1. The output of filter2 is processed by filter3 before being sent to the console window.

A chain must satisfy the following conditions:
- A filter can appear anywhere (at the beginning, between other filters separated by pipes, and at the end), but if it appears at the beginning, the command line must include input

redirection (note: `filter < inFile` is completely equivalent to `type inFile | filter`).

- `MORE` can only appear in the last position of a chain.
- MacDOS commands (with the exception of `MORE`) can only appear at the beginning of a chain.

Note that MacDOS searches the table of internal commands before looking for filters. Therefore, filters named like commands can only be executed if they are preceded by a path.

You should normally be able to get help on MacDOS filters by typing "`help filterName`" ("`filterName/?`" is not supported).

Filters and the Finder

If you launch a filter application by double clicking on it, you can ONLY terminate it by making it the frontmost application and typing `CNTL-C` (or `CMD-dot`). Note that the shutdown will not work, because filters do not accept the standard AEs. This was done to reduce filter code to the minimum.

Speed of filters

You will find that filters are quite slow, especially with less powerful Macs. That is, as soon as you pipe data through a filter, the console window scrolls significantly slower. This is mainly due to the process switches that the Mac OS has to perform before a message goes through all the pipes and back to MacDOS. As a result of this, you will probably decide to use MacDOS pipes only to filter files and commands with a low number of lines or to perform operations in the background while you do something else.

After sending a message to the first filter of a chain, MacDOS expects to receive a message from the last filter. By default, MacDOS reports a timeout error if it fails to receive the message after 2 seconds. You can change this timeout by setting the global variable `TIMEOUT` to the appropriate number of seconds.

Section 12

Getting Info

12 Getting Info

In the previous sections you have already encountered several commands which give you information on your system and MacDOS. `DATE`, `PROMPT`, `SHOW`, `TIME`, and `VOL` all fall into this category. This section completes the list of such commands.

MacDOS Version

You can find out what version of MacDOS you are running by executing the command `VER`. This is part of the information that you get when the Finder displays the Info box of the MacDOS application file.

Additionally, `VER` also displays your MacDOS licence number.

On-line Help

When you type the command `HELP` with a command name as argument, MacDOS displays a brief summary of the command, its parameters, and its switches. You obtain the same result if you execute any command with the switch `/?`.

`HELP` without parameters displays one line of description for each command, while `HELP` followed by a question mark lists the command names.

Keeping Track of MacDOS

You can direct MacDOS to keep a log of the commands it executes. You only need to type: `log logFileSpec`. MacDOS then stores each command that you execute into a new line of the log file. This continues until you execute the command `LOG` without parameters.

Such a file is very useful to trace the execution of complex batch programs, especially during the debugging phase.

The logging mechanism can also be used to generate batch files:

- `LOG` a series of commands;
- edit the log file to remove unnecessary lines and to update some commands;
- change the creator from `'ttxt'` to `'mDOS'`. (you don't need to do this if the system variable `CREATOR` is set to `'mDOS'`).

When used in this way, the `LOG` command operates like the macro collection mechanism common to many applications.

If you add the switch `/O`, `LOG` stores into the log file the output of the commands as well as the commands themselves.

Section 13

Command Reference

13 Command Reference

!, REM

identifies comments (REM stands for the word remark).

Syntax

![*text*]
REM [*text*]

Parameters

text
is one line of text which is ignored by MacDOS. Note that a space is needed after the command name when the REM format is used, while the exclamation mark can be immediately followed by the text.

MacDOS vs DOS

DOS only supports the REM format.

Notes

Input and output redirection
As MacDOS ignores whatever text follows the remark commands, I/O redirection and piping of remarks are not "seen" by MacDOS.

Examples

```
!
REM
REM this is a remark
!*********** another remark ************
! we want to switch off the following command temporarily
!echo off
REMARK: rejected because REM is not followed by a space
```

ALARM

sets MacDOS' way of reporting errors and warnings.

Syntax

alarm [ON | OFF]

Parameters

[ON | OFF]
With ON, ALARM directs MacDOS to report errors and warnings via alerts. When an alert is displayed, execution is suspended until you acknowledge the message. With OFF, ALARM directs MacDOS to report errors and warnings via concise messages. Execution is not suspended. Without parameters, ALARM reports the current setting.

MacDOS vs DOS

DOS does not support ALARM.

Examples

```
ALARM ON
alarm Off
alarm > aFile
```

CALL

starts a program while executing in batch without terminating the current batch program.

Syntax

call *program* [*batch parameters*]

Parameters

program
is the filename of an application, an AppleScript, or a batch program, possibly preceded by an absolute or relative path.

batch parameters
parameters to be passed to the program if it is a batch.

Switches

There are no switches for the CALL command, although some of the parameters for a batch program to be executed may take the form of switches.

MacDOS vs DOS

The two implementations are identical, but MacDOS can execute CALL from the prompt, while DOS only accepts it within batch programs.

Notes

Output redirection
You can redirect the output of batch programs when you CALL them, but CALL does not accept output redirection in any other case.

Recursion
Recursive calls of Batch programs, whether directly or through a chain of batch programs, is certain to cause an error unless you use a terminating condition. This is due to the fact that MacDOS supports a maximum of 16 nested CALLs.

File extension
Batch filenames traditionally terminate with ".BAT", but do not need to. When they do terminate with ".BAT", you do not need to type it. MacDOS first attempts to locate the filename as typed. If the search is unsuccessful, MacDOS then appends the extension ".BAT" and tries again. As MacDOS first tries to find a filename exactly as typed by you, you have to be careful when the same folder contains two files with the same name except for the terminating ".BAT". In that case, you need to type the ".BAT" in order to distinguish between the two files. When MacDOS adds the extension ".BAT", it expects to find a batch file and reports an error in all other cases.

Examples

```
CALL A_BATCH %1 %2 %3
```
The currently executing batch program starts the batch file A_BATCH and passes on to it its own first three parameters.

```
call "Microsoft Word"
```
The currently executing batch program launches MS Word and then continues execution.

Frequently occurring errors

`E27: File or directory not found`
The `CALL`ed program or one of the folders specified in the path could not be found.

CD, CHDIR

change the current folder (ie. DIRectory) or display its name.

Syntax

cd *folder*
chdir *folder*

The two commands are functionally identical.

Parameters

folder
is a name preceded by an absolute or relative path which specifies the new folder. When you use two dots as folder name, `CD` attaches to the folder which contains the current folder. When you use as folder name a volume ID followed by a colon, `CD` displays the current folder of the given volume. Without folder name, `CD` displays the current volume and folder.

MacDOS vs DOS

The two implementations are identical.

Examples

`cd aFolder`
attaches to the folder named `aFolder` and contained in the current folder.

`cd ..`
attaches to the folder which contains the current folder. In DOS, that folder is usually referred to as the "parent directory".

`chdir 1:\`
attaches to the root folder of the start-up volume.

`chdir 2:sub1\sub2`
attaches to the `sub2` folder contained in `sub1`, where `sub1` is a folder contained in the current folder of volume 2.

`cd "\desktop folder"`
attaches to the folder which contains items displayed on the desktop of the current volume.

Frequently occurring errors

`E27: File or directory not found`
The requested folder could not be found. Perhaps it exists but on a different volume.

CLOSE

closes a file opened with the command `OPEN`.

Syntax

close *fileID*

Parameters

fileID

is the file-identification number returned by OPEN.

MacDOS vs DOS

DOS does not support CLOSE.

Notes

Errors

CLOSE does not generate an error message if it fails to close a file. The intent is to encourage you to be on the safe side and perhaps close the files more than once rather than not at all.

Examples

```
close 3
CLOSE %FileID%
```

Frequently occurring errors

```
E13: Syntax: invalid arguments
```

The fileID was not a number. Perhaps you passed a variable name without enclosing it between percent signs, or the variable contained an alphabetic string.

See Also

OPEN, READ

CLS

clears the MacDOS window.

Syntax

cls

MacDOS vs DOS

The two implementations are identical.

CONFIRM

directs MacDOS to ask for confirmation before overwriting or removing existing files.

Syntax

confirm [ON | OFF]

Parameters

[ON | OFF]

CONFIRM ON directs MacDOS to ask confirmation. CONFIRM OFF directs MacDOS to stop asking for confirmation. Without parameters, CONFIRM reports the current setting.

MacDOS vs DOS

DOS does not support CONFIRM.

Examples

```
confirm ON
confirm oFf
CONFIRM >aFile
```

COPY

copies one or more files between locations. Both data and resource forks are copied.

Syntax

copy *source* [,...] [*destination*] [**/A**] [**/C=***creator*] [**/D**] [**/P**] [**/R**] [**/T=***file type*] [**/U**] [**/V**]

Parameters

source [,...]
is a name preceded by an absolute or relative path which specifies the file or group of files to be copied. A group of files is identified via a wildcarded filename or a folder name. To copy several sources, you can specify them one after the other separated by commata.

destination
is a name preceded by an absolute or relative path which specifies the file or folder to which the source[s] are to be copied.

Switches

/A
directs MacDOS to append files to the destination file, so that its initial content is preserved. This switch only takes effect when the destination is an individual file.

/C=*creator*
specifies that only files of a particular creator are to be copied. Note that no spaces are allowed on either side of the equal sign and that the creator string is four characters long. Therefore, if a creator includes spaces, you must double quote the switch (eg. `copy \ "/c=ABC "` copies all the files created by `'ABC '` in the root folder).

/D
specifies that only data forks are to be copied. Nevertheless, when both **/D** and **/R** are ON, MacDOS behaves as if they were both OFF; ie. it copies both forks.

/P
prompts you for confirmation before overwriting existing destination files.

/R
specifies that only resource forks are to be copied. Nevertheless, when both **/D** and **/R** are ON, MacDOS behaves as if they were both OFF; ie. it copies both forks.

/T=*file type*
specifies that only files of a particular type (eg. `TEXT`) are to be copied. Note that no spaces are allowed on either side of the equal sign and that the file type is four characters long. Therefore, if a file type includes spaces, you must double quote the switch (eg. `copy \ "/t=ABC "` copies all the files of type `'ABC '` in the root folder).

/U
specifies that source files overwrite existing destination files only if the source was updated more recently.

/V
re-reads destination files to verify that they have been copied correctly.

MacDOS vs DOS

Source
When you specify several sources in DOS, you must separate them by plus signs. DOS then appends them to each other in a very complex way. With MacDOS, you separate the sources with commata and MacDOS merges them only if you specify a single file as destination.

Destination
Wildcarded destinations are illegal in MacDOS. This limitation is mainly due to the fact that wildcarding is more powerful in MacDOS than in DOS. The matching of source and destination wildcarded identifiers would have become very complex and too difficult to use.

Copying empty files
Unlike DOS, MacDOS copies zero-length files.

Switches
MacDOS provides several switches which are not available in DOS but does not provide the **/A/B** switches to distinguish ASCII and binary copying. MacDOS always performs binary copies.

Merging into one of the sources
When several files are merged into a single file which coincides with one of the sources, MacDOS overwrites that source with the result of the merging. Unlike DOS, MacDOS performs correctly regardless of whether the destination file coincides with the first source or with one of the following sources.

Updating the date/time
In DOS, the command "`copy source+,,`" assigns the current date/time to a file. With MacDOS, you can achieve the same result through the switch **`/D`** of the `RENAME` command.

Notes

Commata between source files
The comma is only allowed between source files. Therefore, a list of sources must not terminate with a comma.

Creators, file types, and finder flags
When copying a file, MacDOS assigns to the destination the same creator, file type, and finder flags. This is also true when several files are merged into a single destination file, but only if all individual source files to be merged have identical parameters. When this is not the case, the parameters which are not identical are replaced by defaults. The default for both creator and file type is `????`, while the default for the finder flags is all-zeroes. For example, when several text files with different creators are merged into a single file, the new file has a `????` creator but is of type `TEXT`.

Dates
When copying files, MacDOS assigns to the destinations the same creation and modify dates of the corresponding sources. When several files are merged into one destination, MacDOS ignores the source dates and sets the destination date to the current date and time on copy completion.

The `/U` switch
You can use the switch **`/U`** to implement a simple backup batch program, as only files updated after the last `COPY` are actually copied.

Order of copying
To know the order in which groups of files are copied, execute a `DIR` command with no sorting options. This might be important when several files are merged into one.

Copying of resource forks
Note that MacDOS **does not** merge resource forks when concatenating files. Therefore, it really only makes sense to append data forks. If you do concatenate files which have resource forks, the resource forks appended to the first one will not be accessible via ResEdit or similar editors.

Examples

In all the following examples, \Dir1 contains: "FILE1.TXT", "FILE1 TXT", "FILE2 TXT", "FI TXT", "FI BLA", "FILE1 BLA", "FILE3 BLA", "FILE1 ZAP", "FILE3 ZAP", and folder "Dir2", while Dir2 is empty. The attach point is Dir2.

```
COPY Dir1\*.*
```
Dir2 then contains FILE1.TXT

```
COPY "Dir1\FILE3 ZAP" Dir1\ZZZ
```
Dir2 remains empty, but a copy of "FILE3 ZAP" named ZZZ is made in Dir1

```
COPY "Dir1\*1 *" ZZZ
```
Dir2 then contains ZZZ, which is the result of concatenating "FILE1 TXT", "FILE1 BLA", and "FILE1 ZAP"

```
COPY DIR1\*3*, Dir1\*2* Z23
```
Dir2 then contains Z23, which is the result of concatenating "FILE3 BLA", "FILE3 ZAP", and "FILE2 TXT"

```
COPY Dir1\*2*,Dir1\*1.* Dir1\FILE1.TXT
```
Dir2 remains empty, but the file "FILE2 TXT" is appended to FILE1.TXT in Dir1.

```
COPY Dir1 Dir2
```
or
```
COPY Dir1\FI* Dir2
```
or
```
COPY ..\*?*?* \Dir1\Dir2
```
Dir2 then contains "FILE1.TXT", "FILE1 TXT", "FILE2 TXT", "FI TXT", "FI BLA", "FILE1 BLA", "FILE3 BLA", "FILE1 ZAP", and "FILE3 ZAP"

Frequently occurring errors

E13: Syntax: invalid arguments
This error is generated in a number of occasions. Check that the destination argument is either a folder name or a filename without wildcards. Also check that a list of source filenames does not end with a comma, as commata are only allowed between source files.

E27: File or directory not found
The source parameter did not identify any file at all or one of the folders specified in the path could not be found. Note that creator and file type are case sensitive. Therefore, /T=text will not find any file of type TEXT.

E28: Bad switch
Perhaps you failed to specify a creator or file type correctly. The equal sign is mandatory and all four characters which form the OSType must be provided.

DATE

displays and sets the system date and the date format. MacDOS uses the format set through DATE whenever it needs to display a date (starting a LOG file, in a DIR listing, etc.).

Syntax

date [*format* | *date*]

Parameters

format
is a string which specifies order and number of digits of year, month, and day. It also specifies the separators to be used between the fields. The default format is yyyy/mm/dd. The characters 'y', 'm', and 'd' used to specify the date format can be in upper or lower case.

date
is the new system date. It is specified in the current format. The Macintosh only accepts dates in the interval 1904 to 2040.

Without parameters, DATE displays the current system date in the current format and prompts you for input. You can reply with either a new date or a new format. If you want to leave both system date and format unchanged, hit the return key.

MacDOS vs DOS

DOS does not let you change the date format via the command DATE.

Notes

Parameter or not parameter
You will probably execute DATE interactively when changing the system date. In that case, execute DATE without parameter, so that MacDOS displays the current settings before prompting you for the new date. This will ensure that you use the correct format when typing the new date.

Format default
Format settings are forgotten when you quit MacDOS. Therefore, it is advisable that you write in autoexec.bat a DATE command which specifies your preferred format.

Formats in input and in output
MacDOS allows some flexibility when a new date is set:

- It ignores leading zeroes. For instance, 00001998, 008, and 0000998 all indicate the year 1998 regardless of whether the year is being displayed as 0001998, 1998, or 0000000001998 (with formats yyyyyyy, yyyy, and yyyyyyyyyyyy respectively).
- It accepts a full value also when its number of digits exceeds the field length. For instance, 1998, 998, 00098, and 008 are all valid inputs regardless of whether the format is yyyyyy, yyy, y, or yyy. This applies to months and days as well, so that October, November, and December can be entered when the length of the month field is a single character.
- It uses the current year as a base for the new year setting. For instance, if the current year is 1998, some of the possible inputs to change it to 1999 are: 0001999, 09, 99, 1999, 00999, and 0000099.

The bottom line is that you can choose the output format for dates without imposing too many constraints in input: as long as you type the correct number of fields in the correct order and with the correct separators, you will probably be ok.

Examples

The following examples illustrate how to change the date format. The 5th day of December 1949 (my birthday!) is used to show the result of the change.

```
date mm/dd/yy          12/05/49
date dd/MM/yyyy        05/12/1949
date yy-m;ddddd        49-2;00005
date yyYYyyy/dd        0001949/05
date yyy               949
date dd mmKyy          05 12K49
date YY/               49
```

As you can see, you can mix and match year, month, and day as you please. Separators must be present and must be single characters but they can be different from each other. The characters yYmMdD0123456789 cannot be used as separators.
Some invalid date formats are:

mm/dd//yy	double separator
mmJJdd	double separator
yyyyDmm	'D' is not a valid separator
mm1dd2yy	'1' and '2' are not valid separators

When setting the system date, you have to follow the current format, but leading zeroes can be omitted.
`date 10/11/12` sets completely different dates depending on the format and the current date. If the current date is January 1st 1993, here are some possible outcomes:

format: `mm/dd/yy` date: 11th October 1912
format: `dd/mm/yy` date: 10th November 1912
format: `yy/mm/dd` date: 11th December 1910

If the current date is January 1st 2001, the same formats result in the following dates:

format: `mm/dd/yy` date: 11th October 2012
format: `dd/mm/yy` date: 10th November 2012
format: `yy/mm/dd` date: 11th December 2010

This is due to the fact that MacDOS adds the leading digits of the year on the basis of the current year.

The following examples show some valid combinations of date formats and date inputs:

format	current date	date input	resulting date
yyyy	16 Aug 1993	4	16 Aug 1994
y/m/d	20 Dec 2001	5/4/3	03 Apr 2005
mmmm/dd	07 Mar 1998	1/1	01 Jan 1998
dd-mm-yy	07 Mar 1998	7-3-2000	07 Mar 2000

And here are some examples of date inputs which would be rejected:

format	current date	date input	why
yyyy	16 Aug 2000	41	year > 2040
y/m/d	20 Dec 2001	5/4/31	April has 30 days
yy	29 Feb 1996	8	1998 not a leap year
dd;mm;yy	13 Mar 1997	14;3	year is missing
dd;mm;yy	13 Mar 1997	14;;98	month is missing
mm-dd-yy	13 Mar 1997	0-15-98	day < 1

Frequently occurring errors

`E47: Invalid date or date format`
You probably used the wrong separators or omitted some fields.

See Also

`TIME`

DECR

decrements or shortens values of global variables.

Syntax

decr [**+** | **-**]*var* [**by** {*number* | *string*}]

"**decr** *var*", equivalent to "**decr +***var* **by** 1", decrements by 1 numeric values and removes one character from the end of alphanumeric strings.

Parameters

[**+** | **-**]*var*
is the name of a global variable, possibly preceded by a plus or a minus sign. No space is allowed between the sign and the variable name. `DECR` uses the sign to decide whether to perform string operations at the end or at the beginning of *var*'s value.

number
is a signed integer number. `DECR` uses *number* differently depending on whether *var* contains a numeric value or not:

- When *var* contains a numeric value, DECR replaces it with the value obtained after subtracting *number*.
- When *var* contains an alphanumeric string which cannot be interpreted as a number, DECR removes from it *number* characters. The characters are removed from the beginning if the variable name is preceded by the minus sign, otherwise they are removed from its end. When *number* is negative, DECR actually extends the content of *var* by appending or prepending spaces (depending on the presence of the minus sign before the variable name).

string
is an alphanumeric string which cannot be interpreted as a number. DECR removes *string* from the string contained in *var*. The string is removed from the beginning if the variable name is preceded by the minus sign, otherwise the string is removed from its end.

MacDOS vs DOS

DOS does not support DECR.

Notes

Removing numeric characters
When the BY-string is numeric, DECR decrements a numeric variable by that amount or removes that number of characters from an alphanumeric variable. If your purpose is to remove digits from a variable (eg. the terminating '1' from the string "1341"), you can achieve this result by adding an alphabetic character to the variable before removing the digit (ie. INCR var BY x followed by DECR var BY 1x).

Decrementing too much
When the number of characters that you want to remove from a string is greater than the length of the string, DECR removes all the characters contained in the variable without returning any error. The variable is then automatically removed from the environment if it is user-defined, or reset to its default if it is a system variable.

Examples

initial var	command	resulting var
"25"	decr var	"24"
"25"	decr -var	"24"
"+25"	decr var by 30	"-5"
"25"	decr var by -3	"28"
"-7"	decr var by -30	"23"
"abcde"	decr var	"abcd"
"abcde"	decr -var	"bcde"
"abcde"	decr var by 3	"ab"
"abcde"	decr var by -1	"abcde "
"abcde"	decr -var by -3	" abcde"
"ab de"	decr var by de	"ab "
"a cde"	decr -var by "a "	"cde"
"abcde"	decr +var by 7	var removed
"abcde"	decr var by abcde	var removed
"abcde"	decr var by fgh	fails

Frequently occurring errors

E63: Requested variable has not been defined
You attempted to DECRement a variable that did not exist. Remember that empty variables are automatically removed unless they are system variables.

E72: String search failed
This error is often due to the fact that DECR is case sensitive. Therefore, the removal of a substring does not succeed if you do not match upper and lower cases sequences. You also get this error if you erroneously typed DECR instead of INCR.

See Also

INCR, SSTR, TOUPPER

DEL, ERASE

remove from the system (ie. DELete) one or more files.

Syntax

del *file* [**/C=***creator*] [**/P**] [**/T=***file type*]
erase *file* [**/C=***creator*] [**/P**] [**/T=***file type*]

The two commands are functionally identical.

Parameters

file
is a name preceded by an absolute or relative path which specifies the file or group of files to be deleted. A group of files is identified via a wildcarded filename or a folder name.

Switches

/C=*creator*
specifies that only files of a particular creator are to be deleted. Note that no spaces are allowed on either side of the equal sign and that the creator string is four characters long. Therefore, if a creator includes spaces, you must double quote the switch (eg. `del \ "/c=ABC "` deletes all the files created by `'ABC '` in the root folder).

/P
prompts you for confirmation before deleting each file.

/T=*file type*
specifies that only files of a particular type (eg. TEXT) are to be deleted. Note that no spaces are allowed on either side of the equal sign and that the file type is four characters long. Therefore, if a file type includes spaces, you must double quote the switch (eg. `del \ "/t=ABC "` deletes all the files of type `'ABC '` in the root folder).

MacDOS vs DOS

The two implementations are identical.

Notes

Deleting everything
Under certain circumstances, MacDOS prompts you for confirmation before beginning execution of the DEL command. This happens when you type one of the following three commands:

- `DEL *`
  to delete all the files in the current folder.
- `DEL *.*`
  to delete all the files of the current folder which have a DOS-like name. This particular case has been copied from DOS because DOS users would expect to be prompted.
- `DEL foldname`
  to delete all the files in the folder named `foldname`.

MacDOS suppresses this prompt while executing in batch or when you request to be prompted before deletion of each individual file (CONFIRM is ON or the DEL option /P is ON).

Deleting files vs removing folders
DEL only operates on files. Therefore, when you pass to DEL the name of a folder, MacDOS deletes the files contained in the folder rather than the folder itself. If you want to remove a folder, you have to use RMDIR.

Trashing vs DELeting
DELetion is equivalent to moving files to the Trash Can and then emptying the trash. That is, DELeted files are totally removed from the system. If you want to have a behaviour similar to the Macintosh Finder, use the batch program `trash.bat` included as example in the MacDOS package.

Examples

`DEL oldDocs /t=WDBN /p`
deletes all MS-Word documents in the folder named `oldDocs`. It also asks confirmation before deleting each file.

`DEL * /t=TEXT /c=MSWD`
deletes all text files produced with MS-Word in the current folder.

`DEL 1:\Trash`
removes all the files contained in the Trash Can of the start-up volume.

`DEL 2:\*old*`
deletes all the files in the root of volume 2 whose name contains the string "`old`" (watch out; such a command also deletes files like `myGoldenFile`, `FOLDERlist`, `Soldano.txt`, etc. which are not necessarily old).

Frequently occurring errors

`E27: File or directory not found`
The file parameter did not identify any file at all. Perhaps, you asked to delete the contents of a folder but the folder did not contain any file.

See Also

`RD, RMDIR`

DIR

lists the contents of a folder (ie. a DIRectory).

Syntax

dir [*selection*] [**/**[**-**]**A**[[:]*attribute list*]] [**/**[**-**]**B**] [**/C=***creator*] [**/**[**-**]**L**] [**/**[**-**]**O**[[:]*ordering list*]] [**/**[**-**]**P**] [**/**[**-**]**S**] [**/T=***file type*] [**/**[**-**]**W**]

Parameters

selection
is a name preceded by an absolute or relative path which specifies the items to be listed. A group of files is identified via a wildcarded filename or a folder name. If *selection* is omitted, `DIR` lists the contents of the current folder.

Switches

/[**-**]**A**[[:]*attribute list*]
only lists items with attributes included in the *attribute list* (but look at the note concerning `DIRCMD`). The *attribute list* is a string of single letter attributes:

- D for Directories,
- F for visible Files,
- H for Hidden files,
- X for aliases.

A minus sign preceding an attribute tells MacDOS not to list items with that attribute. A minus sign between the `/` and the `A` inverts the logic of all attributes in the *attribute list*. By default, MacDOS does not list hidden files.

/B

lists item names one per line and without headings (bare format). When the **/S** option is also ON, **/B** writes the full path instead of the item name only. A minus sign between the / and the B switches this option OFF.

/C=*creator*
specifies that only files of a particular creator are to be listed. Note that no spaces are allowed on either side of the equal sign and that the creator string is four characters long. Therefore, if a creator includes spaces, you must double quote the switch (eg. `dir \ "/c=ABC "` lists all the files created by `'ABC '` in the root folder).

/L
operates case-sensitive matching on the *selection*. A minus sign between the / and the L switches this option OFF.

[**/**[**-**]**O**[[**:**]*ordering list*]]
sorts the items as specified in the *ordering list* before listing them. The *ordering list* is a string of single letter sorting criteria:

D	by Date & time of update (older first)
G	Group directories first
N	by Name (alphabetic)
S	by Size (smallest first)

A minus sign preceding a sorting criterion tells MacDOS to sort in the reverse order. A minus sign between the / and the O inverts the logic of all criteria in the *ordering list*.

/P
pauses after each screenful of listing. A minus sign between the / and the P switches this option OFF.

/S
provides a recursive display of the items in all subfolders. A minus sign between the / and the S switches this option OFF.

/T=*file type*
specifies that only files of a particular type (eg. TEXT) are to be listed. Note that no spaces are allowed on either side of the equal sign and that the file type is four characters long. Therefore, if a file type includes spaces, you must double quote the switch (eg. `dir \ "/t=ABC "` lists all the files of type `'ABC '` in the root folder).

/W
lists item names in a wide format, with as many items per line as possible. A minus sign between the / and the W switches this option OFF.

MacDOS vs DOS

Switches
The switches /AS, /AA, /AR, and /OE are not implemented, but /AF, /AX, /C, and /T have been introduced.

Directory header
To cope with the long names typical of the Mac, the path in the directory header is displayed with one directory name per line.

Names containing spaces
All names displayed are quoted, so that leading and trailing spaces are made visible. Use the system variable QUOTE to change the quoting character.

Creators and File Types
Creator and file-type are displayed.

Listing of aliases
Target volume and target filename of aliased items are displayed immediately after the directory entry for the alias file itself.

Date and time formats
Date and time are displayed in the format chosen by you (refer to the two commands `DATE` and `TIME`).

Notes

`DIRCMD`
Before executing the command `DIR`, MacDOS selects the switches contained in the system variable `DIRCMD`. Therefore, `DIRCMD` represents the system default switches. You can override the defaults by explicitly including switches in the command (`/-B` negates `/B`, `/W` negates `/-W`, etc.). Alternatively, you can `SET DIRCMD` to your own preferred defaults. For instance, you can include `/AH` so that hidden files are always displayed. By including such a `SET` command in the `autoexec.bat` file, you only need to type your preferred defaults once. The system defaults are:

`/-P`	do not pause at the end of each page
`/-W`	do not list in wide format
`/A:-HDXF`	list directories, aliases, and files, but not hidden files
`/-O`	do not sort
`/-S`	list only the current folder (not recursively)
`/-B`	do not list in bare format
`/-L`	operate case-insensitive

Repetition of switches in the same command
Switches can be repeated. As MacDOS scans the switches from left to right, the rightmost switch of a particular type overrides the preceding ones. `/A` and `/O` can be repeated to extend their sub-option lists. For example, `/A-H-D` is totally equivalent to `/A-H/A-D`.

Conflicting attributes in the `attribute list`
MacDOS scans the attributes of the `/A` switch from left to right. Therefore, the rightmost attribute of a particular type takes effect. For example, `/A:HDF-H` does not display hidden files. If you type `/-ADFH` (or an equivalent sequence of attributes) you never get any item listed, because you exclude both folders and files (aliases are particular cases of files).

Complex sorting
MacDOS scans the ordering list from left to right. Therefore, the rightmost sorting is done last. For example, `/O:SN` first sorts by size, and then by name. Note that names are unique. Therefore, if you sort by name, all preceding sorts become totally irrelevant. A similar consideration also applies to `/O:D`, because the time of update is also almost always unique. The situation is different when you apply `/ON` or `/OD` before applying `/OG` or `/OS`. For example, if you apply `/O:NSG`, you list the directories ordered by name followed by the files ordered by size, with files of the same size in alphabetical order of their names.

Sorting and recursion
Each folder is sorted individually.

Aliases
Under normal circumstances, `DIR` displays two lines for each alias. The first line refers to the alias file itself, while the second line refers to the target. If the alias cannot be resolved (perhaps because the target is on a volume which has not been mounted), `DIR` reports a warning and then only displays the first line with an indication that the file is an alias.

Note that when listing an alias, MacDOS always lists the corresponding targets, regardless of their type and visibility.

Memory limitations
DIR saves folder information in dynamically allocated memory rather than on disk. This speeds up the listing but limits the number of directory items which MacDOS can handle. With the current memory partition, this limit is of approximately 1100 entries. If MacDOS returns the error "Not enough space in the heap", the only solution (short of splitting the folder which is causing problems) is to increase the amount of memory allocated to MacDOS. To do so, exit MacDOS, click once on the application icon, press cmd-I, and increment the partition size.

Examples

`dir`	lists the contents of the current folder.
`dir 2:/p`	lists the contents of the current folder of volume 2 and pauses at the end of each page.
`dir abc`	lists the file `abc` or the contents of the folder `abc` contained in the current folder
`dir ..\abc*`	lists all the items of the parent folder whose name begins with `abc`.
`dir \/w`	lists the contents of the root folder in a wide format.
`dir/t=TEXT/s`	recursively lists all text files in the current folder and in its subfolders
`dir \a\b\*TEMP*/L`	lists all the files in the folder `\a\b` whose name contains the string `TEMP` (case sensitive).
`dir/o-s/a-d`	lists the files (not the folders) in the current folder in order of size (from the largest to the smallest).
`dir \/s/b/c=ABCD/a-d`	lists all the files on the current volume created by `ABCD`. For each item, the full path and name is given.
`dir \t*/s/b/a-f`	lists all the folder names in the current volume which start with '`t`' or '`T`'.
`dir/a-d-x/s/b>aFile \`	saves into a file named `aFile` the list of all non-alias files in the current volume. For each item, the full path and name is given.
`dir /o-d/c=MWII`	lists all MacWrite II documents in order of update, from the newest to the oldest.
`dir 1:\/s/b/a-d-fh`	lists all hidden files on the startup volume. For each item, the full path and name is given.

Frequently occurring errors

`E27: File or directory not found`
Perhaps you specified a proper filename (without wildcards) but the file does not exist. Alternatively, it is one of the folders within the path which does not exist.

See Also

`TREE`

ECHO

displays messages or turns batch command echoing `ON` and `OFF`.

Syntax

`echo` [{*`message`* | **`ON`** | **`OFF`**}]
`echo.`

Parameters

{*`message`* | **`ON`** | **`OFF`**}
is a string to be displayed in the MacDOS window. If the string only consists of the word `ON`, `ECHO` does not display it but enables the echoing of commands to the MacDOS window. Similarly, `OFF` causes the echoing to be disabled. If the parameter is entirely missing, `ECHO` displays the current echo setting (`ON` or `OFF`).

ECHO immediately followed by a period (a dot) just displays an empty line.

MacDOS vs DOS

The two implementations are identical.

Notes

Interactive usage

MacDOS lets you use ECHO interactively. Therefore, if you want to view a single global variable, type ECHO %var% rather than SET .

Output redirection

ECHO redirects its output when you include it in a batch program and redirect the output of the whole batch.

EJECT

dismounts and places off-line removable volumes.

Syntax

eject *volume* [**/E**]

Parameters

volume

is the ID which identifies the volume to be dismounted or placed off-line. Without switches, EJECT dismounts the volume.

Switches

/E

specifies that the volume is only to be placed off-line (eg. Ejected) without being dismounted.

MacDOS vs DOS

DOS does not support EJECT.

Notes

The start-up volume cannot be dismounted regardless of whether it is removable or not.

Examples

```
eject 2:
```

removes the volume with ID 2. The icon disappears from the desktop.

```
eject D /e
```

ejects the volume D. The icon is then greyed.

EXIT

quits MacDOS, other applications, or the Finder.

Syntax

exit [*appl***]**
exit finder

Parameters

appl

identifies the application that should receive a Quit Apple Event. If the parameter is missing, MacDOS terminates itself.

MacDOS vs DOS

In DOS, EXIT is used to quit the current shell and possibly return to a program which was suspended.

Notes

EXIT's main purpose is to use batch programs as stand-alone applications which automatically terminate after completing their task.

Quitting the Finder

When you kill the Finder, you no longer have access to the items normally in the Apple menu. Be warned that MacDOS relies on the 'MACS' creator to identify the process associated with the Finder. If you have already replaced the Finder with a different application, the "exit finder" command will have no effect.

You can re-launch the Finder by typing its path/filename as if it were a normal application. Make an alias of the Finder, name it F, and place it in a folder included in the search PATH. You will then be able to then re-launch the Finder with just two key strokes.

FOR

executes a command or a series of commands for each file in a set.

Syntax

for %*var* [**/L**] **in (***set***) do** [*command* | **begin**]

Parameters

var

is the name of the variable that FOR uses to store the names of the files in *set*, one at a time. When you use the format "**do** *command*", type the string %var where you would normally type a filename. FOR then executes the command for all files in the set, one at a time. When you use the format "**do begin**", *var* becomes a global variable rather than a variable local to FOR. Therefore, type %var% to obtain the name of the current file.

set

is a list of space-separated items, where each item is either a file or a folder name. Each item can be preceded by a volume and path spec, and filenames can be wildcarded. The parentheses are mandatory.

command | **begin**

is either the single command to be executed or a keyword which starts the execution of a series of command lines. With **begin**, MacDOS keeps executing command lines until it encounters a line containing the command NEXT var . At that point, MacDOS sets *var* to the next filename in the set and resumes execution from the line immediately following the **begin**. After 'using' all the files in *set*, MacDOS continues execution of the batch program from the line immediately following the NEXT.

Switches

/L

leaves the filenames in the set as they are, without converting them to uppercase. The default is that the filenames are converted. This switch must precede the **do**.

MacDOS vs DOS

Multi-line FORs

The keyword **begin** allows for loops spanning over more than one line.

De-referencing the loop control variable
With standard single-line FORs, the control variable is always de-referenced by prepending a single percent sign (as in %varName), regardless of whether the FOR is in a batch program or not. DOS needs a double percent sign in batch programs. With multi-line loops, the control variable becomes a standard global variable and is de-referenced normally (as in %varName%).

Lower case option
DOS does not support /L.

What can be done
Beside normal commands, MacDOS also accepts program names and the command CALL after the **do**.

Folder expansion
Folder names in a set are expanded, so that (folderName) in MacDOS is equivalent to (folderName*.*) in DOS. This seems more consistent with the way in which folder names are usually handled by file-oriented commands.

Notes

Replacing FOR
FOR is a command which has mainly been implemented to help experienced DOS users. Although it has been extended in a series of ways, its use remains limited because it does not distinguish between file types, does not return full paths, and does not operate recursively. The most flexible way of operating on a set of files with MacDOS is to use the following strategy:

- redirect the output of DIR/B to a file (all the filtering options of DIR as well as its recursive operation are available);
- OPEN the file;
- repeat the READing of the file, so that a filename at a time is stored in a global variable and is available for processing;
- CLOSE the file (only if you do not reach the EOF, otherwise MacDOS closes the file automatically);
- reSET the global variables used and DELete the file (always clean-up the mess before you start doing something else).

Interactive vs. batch
You can execute the standard FOR interactively, while you can only use the multi-line FOR within batch programs.

What happens to the control variable after exiting the FOR loop
The control variable of standard single-line FORs is a local variable and is not accessible within other commands. On the other hand, the control variable of multi-line FORs is a normal global variable. After terminating the loop, such a variable contains the name of the last file in the set and it is your responsibility to remove it by SETting it to the empty string. To remove it immediately after exiting the loop is a good defensive practice.

I/O redirection
Redirection is only legal with the standard single-line format of FOR. In that case, it applies to the command which follows the **do** rather than to the FOR itself.

Piping of the output to other commands is forbidden except to MORE. To put it simply, FOR only supports piping in the following case:

```
FOR %var [/L] IN (set) DO internal-com-with-params | MORE
```

Incompatibility of commands

Standard FORs cannot be nested. That is, no FOR command is allowed after the **do** keyword of another FOR. IFs are also forbidden within standard FORs.

Empty set
If the set does not select any file, FORs do not return any error and multi-line FORs do not create the global variable.

Running away
If a command within a FOR generates a file which belongs to the set, a situation in which the loop "feeds" itself with new files can occur. Such a "runaway" condition would only stop upon hitting some system boundaries. Here are a couple of examples:

```
for %var in (d*) do dir/b > %varX
```
this command generates a file for each file whose name begins with the letter 'd'. Unfortunately, the new file also begins with 'd' and is therefore part of the set. To see what happens, let's say that you have a file named DATA. In the first iteration of the FOR-loop, MacDOS generates DATAX; in the second iteration DATAXX; then DATAXXX; etc. This only stops when the maximum length of filenames (31 characters) is reached. If you had named the new files X%var instead of %varX, you would have been fine.

```
set counter=0
for %var in (*t*) do begin
     incr counter
     copy %var% data%counter%
next var
```
here the purpose was to make copies of all files containing the letter 't' and name them data1, data2, etc. Again, the problem is that the new files are in the set. New files are probably created until the hard disk is full. You could have named the new files xxx%counter% and then renamed them with `ren xxx* data*` at a later stage. Alternatively, you could have stored the copies into a different folder.

The general solution is to replace FOR with a combination of DIR and a loop. This would avoid the risk of "positive feedback".

Examples

```
for %a in (wildcardedFilename) do more < %a
```
executes a MORE on all files identified by wildcardedFilename (it only works with files in the current folder and if they are of type 'TEXT').

```
for %a in (myDir) do more < myDir\%a
```
executes a MORE on all files in the folder myDir (it only works if the files are of type 'TEXT').

Frequently occurring errors

E27: File or directory not found
At least one of the paths specified in the set included a folder which does not exist.

See Also

NEXT

GOTO

while in batch, continues execution from a labelled line.

Syntax

goto *label*

Parameters

label

is an alphanumeric string used as a label in the current batch program. Remember that the colon which identifies the label is not part of the label itself.

MacDOS vs DOS

The two implementations are identical.

Examples

```
goto QUIT_LBL
goto 4
GOTO lbl10
goto %var%
```

Frequently occurring errors

`E44: Label not found`
The requested label was not defined in the current batch program.

HELP

provides on-line information for all MacDOS commands.

Syntax

help [*entry*]

Parameters

entry
is either a command name, a file specification, or a question mark.

When *entry* is a command name, `HELP` displays a description of the command.

When *entry* identifies a file, `HELP` searches the resource fork of the file looking for a resource of type `'TEXT'` named "help". If it finds it, it display its contents in the console window. This lets you get help on any program, filter, and script provided it contains the appropriate `'TEXT'` resource.

When *entry* is a question mark, `HELP` lists all the command names.

If *entry* is missing, `HELP` lists brief descriptions of all the commands.

MacDOS vs DOS

The two implementations are identical, except for the fact that MacDOS supports `"HELP ?"` and can read "help" text resources.

IF

performs conditional processing in batch.

Syntax

if [**not**] {*s1* **==** *s1* | **exist** *name* | **existdir** *name*} *command*

Parameters

{*s1* **==** *s1* | **exist** *name* | **existdir** *name*}
is a condition which can have three forms:

s1 == s1
is true when the two strings *s1* and *s2* are identical. The two strings can contain replaceable parameters and global variables.

exist *name*
is true when a file named *name* exists. The actual filename can be preceded by a volume specification and path.
existdir *name*
is true when a folder named *name* exists. The actual folder name can be preceded by a volume specification and path.

A **not** preceding a condition changes a true condition to false and vice versa.

command
is executed when the condition is true. It can be a MacDOS command or the name of an executable program.

MacDOS vs DOS

Keywords
DOS does not support **existdir**; MacDOS does not support **errorlevel**.

Hidden files
Hidden files **exist** for MacDOS but not for DOS.

Notes

Aliases
When the item tested with **exist** or **existdir** includes a path, MacDOS resolves all the aliases of folders in the path but not the innermost one. Therefore, aliases always satisfy **exist** and never **existdir**, regardless of whether the target is a file or a folder.

Incompatibility with FOR
IFs are not allowed within standard FORs. That is, you cannot type:

```
for %var in (set) do if %var == ...
```

The solution is to use multi-line FORs (or to drop FORs altogether). Note that you can execute a FOR within an IF.

Reserved words
The words **not**, **exist**, and **existdir** are reserved in both upper and lower case. Therefore, commands like `if not == %var%` are in most cases rejected with a syntax error. Here is an example in which IF might not do what you expect:

```
if not exist == %1 do_this
```

The purpose is probably to execute do_this if the first replaceable parameter contains the string "exist". Unfortunately, IF interprets the command line as a request to execute "%1 do_this" if a file named "==" does not exist. Most likely, you do not have such a file. Therefore, MacDOS will attempt to execute "%1 do_this". This can be very confusing, especially when you have ECHO disabled. To avoid this kind of problems, you have the following choices:

- Never use reserved words (preferred solution).
- Double-quote reserved words and variables which could contain reserved words:

```
if "%var%" == "not" do_this
if not "exist" == %1 do_this
```

Spaces
Spaces within a condition are ignored, unless you double-quote them. Also, strings containing spaces must be double quoted.

Examples

Check the existence of a file:

```
if exist %1 echo file "%1" exists
```

Check the non-existence of a folder:

```
if not existdir %1 echo folder "%1" not there
```

Use a batch program to process all files in a folder, but only after checking that the folder exists:

```
if existdir %1 for %file in (%1) do processFile %1\%file
```

Check the control variable of a loop:

```
if %counter% == 7 goto EXIT_LOOP_LBL
```

Check an error condition:

```
if not %doserr% == 0 goto ERROR_LBL
```

Frequently occurring errors

`E13: Syntax: invalid arguments`

Very often this error message is caused by a reference to a replaceable parameter which was not defined. For example, `if %1 == 3 do_this` fails if `%1` is empty. It is so because MacDOS, after replacing `%1`, only sees `if == 3 do_this` and cannot make any sense of it. To avoid this problem, append a character to both sides of the condition (a space does not work unless you use double quotes): `if %1x == 3x do_this`. Then MacDOS sees `if x == 3x do_this` and behaves as expected.

INCR

increments or extends values of global variables.

Syntax

incr [**+** | **−**]*var* [**by** {*number* | *string*}]

"**incr** *var*", equivalent to "**incr +***var* **by** 1", increments by 1 numeric values and appends one space to the end of alphanumeric strings.

Parameters

[**+** | **−**]*var*

is the name of a global variable, possibly preceded by a plus or a minus sign. No space is allowed between the sign and the variable name. INCR uses the sign to decide whether to perform string operations at the end or at the beginning of *var*'s value.

number

is a signed integer number. INCR uses *number* differently depending on whether *var* contains a numeric value or not:

- When *var* contains a numeric value, INCR replaces it with the value obtained after adding *number*.
- When *var* contains an alphanumeric string which cannot be interpreted as a number, INCR extends it by adding *number* spaces. The characters are inserted before the beginning if the variable name is preceded by the minus sign, otherwise they are appended after its end. When *number* is negative, INCR actually shortens the content of *var* by removing characters from the end or from the beginning (depending on the presence of the minus sign before the variable name).

string

is an alphanumeric string which cannot be interpreted as a number. INCR joins *string* to the string contained in *var*. The string is prepended if the variable name is preceded by the minus sign, otherwise the string is appended.

MacDOS vs DOS

DOS does not support INCR.

Notes

Appending numeric characters

When the BY-string is numeric, INCR increments a numeric variable by that amount or adds that number of spaces to an alphanumeric variable. If your purpose is to add digits to a variable (eg. appending `'1'` to the string `"134"` so that it becomes `"1341"`), you can achieve this result by

extending the BY-string with an alphabetic character and then immediately removing it from the variable (ie. INCR var BY 1x followed by DECR var BY x).

Examples

initial var	command	resulting var
"25"	incr var	"26"
"25"	incr -var	"26"
"-25"	incr var by 30	"5"
"25"	incr var by -3	"22"
"-7"	incr var by -30	"-37"
"abcde"	incr var	"abcde "
"abcde"	incr -var	" abcde"
"abcde"	incr var by 3	"abcde "
"abcde"	incr var by -1	"abcd"
"abcde"	incr -var by -3	"de"
"ab de"	incr var by de	"ab dede"
"a cde"	incr -var by "a "	"a a cde"
"abcde"	incr +var by -7	var removed

Frequently occurring errors

E63: Requested variable has not been defined
You attempted to INCRement a variable that did not exist. Remember that empty variables are automatically removed unless they are system variables.

See Also

DECR, SSTR, TOUPPER

LOG

captures MacDOS commands and their output.

Syntax

log [[*file*] [**/A**] [**/O**] | [**/-**]]

Parameters

file
is the file to be used for storing captured information. It can include a volume and path spec. If *file* is missing, LOG closes the current logging file and stops logging.

Switches

/A
appends logging data to the requested file instead of overwriting it.

/O
captures the output of commands. When /O is missing, only the commands themselves are captured.

/-
suppresses the storing of the closing message when closing a log file. You can then use a log file as a batch program without having to remove the closing message by hand. It only makes sense if the log file was opened WITHOUT the /O option.

MacDOS vs DOS

DOS does not support LOG.

Notes

File creator

By default, LOG generates a file with creator 'ttxt' (for TeachText), but you can change it by SETting the system variable CREATOR.

Frequently occurring errors

E77: File already open for writing
The file was already open in exclusive mode. Perhaps you were already using the same file for LOGging.

MD, MKDIR

create a folder (ie. MaKe a DIRectory).

Syntax

md *name*
mkdir *name*

The two commands are functionally identical.

Parameters

name
is the name of the folder to be created. It can include a volume and path spec.

MacDOS vs DOS

The two implementations are identical.

Examples

Create a folder in the root directory:

```
mkdir \foldName
```

Create a subfolder of the current folder:

```
mkdir subFold
```

Create a folder in the same folder which contains the current one:

```
mkdir ..\sisterFold
```

Create a folder in the current folder of volume D:

```
mkdir D:newFold
```

Frequently occurring errors

E58: Duplicate filename(s)
There is already a folder or a file with the same name in the same place. Remember that names are NOT case sensitive.

See Also

RD, RMDIR, CD, CHDIR

MEM

provides information on the list of executing processes.

Syntax

mem

MacDOS vs DOS

MacDOS provides information similar to what is provided in DOS by the command MEM /PROGRAM. There are two main differences:

- MacDOS does not provide information on system globals.

- MacDOS provides additional information like the creator string and the path of the application file.

MORE

displays the content of a text file one page (ie. one window-full) at a time.

Syntax

more < *file*
chain | **more**

Parameters

file
is the name of a text file, possibly preceded by a volume and path spec. When *file* is missing, MORE accepts input from the keyboard. Input can then be terminated by typing a cntl-Z (that the Macintosh interprets as an EOF), a cntl-C, or a cmd-dot.

chain
is a chain of piped MacDOS filters, possibly including parameters and switches and beginning with a command. In its simplest form, a *chain* simply consists of a MacDOS command that accepts output redirection.

MacDOS vs DOS

MacDOS does not prompt the user at the end of each page when the output is redirected to a disk file.

DOS only allow MORE to be preceded by a single command.

Notes

Generating text files
You can generate small text files by executing MORE without parameters by redirecting its output to a disk file:

```
more > newFile
```

MORE then keeps storing what you type into newFile until you type a cntl-Z. If your keyboard does not have the CNTL key, you can type cmd-dot instead. Nevertheless, as a cmd-dot aborts the current line, you should only type it after the last RETURN.
This method of producing text files is only practical for batch programs of a couple of lines or small files for test purposes.

Examples

```
more < \documents\myTextFile
myFilter < filterInput | more
dir | more
help xcopy | more
dir/b/a-d | filter1 | filter2 | more
```

Frequently occurring errors

```
E40: Not a file of type 'TEXT'
```

MORE can only display text files.

See Also

TYPE

NEXT

terminates a multi-line FOR-loop.

Syntax

next *varname*

Parameters

varname
is the control variable of the FOR-loop.

MacDOS vs DOS

DOS does not support NEXT.

Notes

Jumping in and out of FORs
If you jump into a FOR-loop with a GOTO, MacDOS does not automatically report an error. This is partly due to the fact that MacDOS ignores the NEXT command if the control variable does not exist. You should always enter loops from the top. If a FOR-loop is exited with a GOTO before it is completed, MacDOS "remembers" that the loop was not completed and does not release the control variable. Therefore, it is possible to jump out, do something, and jump back in.
If you jump out of a FOR-loop and then try to restart it from the beginning, MacDOS reports the error message:

```
Variables used in iterations must be unique
```

because the control variable was not released.

See Also

FOR

ONERROR

specifies a label for error handling in batch.

Syntax

onerror [*label*]

Parameters

label
is a label from which batch execution continues whenever MacDOS encounters an error. When this parameter is missing, ONERROR resets the error handling.

MacDOS vs DOS

DOS does not support ONERROR.

Notes

Standard handling
After detecting an error, MacDOS aborts the current command and continues execution of the batch program from the following line.

Error routine
You can implement an error routine by setting a variable with a return label:

```
onerror ERR_LBL
set return_lbl=RET_LBL
```

if this line causes an error we resume execution from ERR_LBL

```
:RET_LBL
```

here is where we come back from the error routine

```
    ...
:ERR_LBL
    here we handle the error before resuming normal execution
    goto %return_lbl%
```

Examples

```
onerror
onerror ERR_LBL
onerror %var%
```

See Also

SHOW

OPEN

opens a text file for sequential access.

Syntax

open [*file* [*var*]] [{**/R** | **/W** | **/A**}]

Parameters

file
is the name of a file of type 'TEXT', possibly preceded by a volume and path spec. When *file* is missing, OPEN displays the list of files already open. After being OPENed, the file can be accessed with the commands READ or WRITE and closed with CLOSE.

var
is the name of the variable into which the fileID is to be stored. When *var* is missing, OPEN displays the fileID in the MacDOS window.

Switches

Only one of **/R**, **/W**, and **/A** at a time is allowed in a command. When no switch is provided, **/R** is assumed.

/R
opens an existing text file to read (see the command READ). If the file does not exist, OPEN fails.

/W
opens a new text file to write (see the command WRITE). If the file already exists, OPEN replaces it with the new one.

/A
opens an existing text file to write past the current EOF (see the command WRITE). If the file does not exist, OPEN creates it.

MacDOS vs DOS

DOS does not support OPEN.

Notes

Forks
OPEN only sees the data fork of text files.

Non-text files
You can READ the data fork of a file not of type 'TEXT' by copying it:

```
COPY/D filename theCopy
```

and changing its type:

```
REN theCopy /t!TEXT
```

(at this point, you can also use TYPE and MORE to display it).

Concurrent access
The same file can be opened more than once for reading but only once for writing. This also applies to files opened directly by MacDOS (eg. to read a batch program or for I/O redirection) or by other applications (compilers, word processors, etc.). You should not change the content of a file while it is being read. When you OPEN a file, MacDOS keeps in memory the current file offset and has no way of checking whether it remains valid. Therefore, it is of particular importance that the length of the portion already READ remains unchanged.

File reset
If you OPEN a file for writing and CLOSE it immediately thereafter, this will have the effect of removing its content, while leaving its creator unchanged.

Examples

```
open myFile
open 2:\aFold\aFile aVar /R
open memoList saveID /a
open
```

Frequently occurring errors

`E27: File or directory not found`
When OPENing for reading, the parameter did not identify any file at all. Alternatively, one of the directories specified in the path could not be found.

`E34: Illegal wildcarding`
You were trying to open a folder instead of a file: MacDOS tried to open all the files contained in the folder and then failed because OPEN can only open a file at a time.

`E40: Not a file of type 'TEXT'`
You can only OPEN text files. Perhaps you forgot to save it from your word processor as "text only".

`E77: File already open for writing`
The file to be OPENed for writing was already open in exclusive mode, either by MacDOS or by another application. Check what files you have opened in your word processor and/or editor. Some applications fail to close a file when you close the corresponding window (yes, it is a bug!). In those cases, you will have to quit the application in order to free the file.

See Also

READ, WRITE, CLOSE

PATH

displays and sets search paths for executable files.

Syntax

path [*paths*]

"PATH paths" is equivalent to "SET PATH=paths"

Parameters

paths
is a list of paths separated by semicolons. If *paths* is missing, PATH displays the current list. If *paths* consists of a single semicolon, PATH resets the list.

MacDOS vs DOS

The two implementations are identical.

Notes

Global variable PATH

MacDOS stores the list of paths in a global system variable named PATH. Therefore, the command

```
path listOfPaths
```

is completely equivalent to

```
set path=listOfPaths
```

Similarly,

```
path ;
```

is equivalent to

```
set path=
```

Changing the current PATH

You can easily append a new folder to the current PATH by typing any of the following three commands:

```
path %path%;newFolder
set path=%path%;newFolder
incr path by ;newFolder
```

Similarly, to insert a new folder at the beginning of the current PATH, you can type:

```
path newFolder;%path%
set path=newFolder;%path%
incr –path by newFolder;
```

Also look at the batch programs addPath.bat and delPath.bat in the batches folder of the MacDOS floppy.

Order of searching and multiple filenames

MacDOS looks for executable files in the current folder (first without extension, then after appending the extensions .BAT) before scanning the folders identified in the global variable PATH.

When looking for an executable file or a file of abbreviations, MacDOS scans the paths from left to right until it finds a file with the correct name. Therefore, when several files with the same name are in the different folders of the path, MacDOS only sees the first one.

Examples

```
path ;
path
path "c:\;\bin;1:\application folder;1:\batches"
```

PAUSE

suspends execution of a batch program and asks whether execution should be resumed.

Syntax

pause

MacDOS vs DOS

The two implementations are identical.

Notes

PAUSE vs cntl-C

PAUSE lets you interrupt a batch program in a controlled way, while a cntl-C (or cmd-dot) interrupts batch execution asynchronously. After being PAUSEd, a batch program can always continue without problems. On the other hand, cntl-C is risky, because it actually interrupts individual commands. When you resume execution after a cntl-C, MacDOS does NOT complete the command which was interrupted, but simply continues with the next line of the batch program.

PRINT

prints a text file.

Syntax

print [*file*] [**/P**] [**/D**]

Parameters

file

specifies what is to be printed. It can be a filename, a wildcarded filename, or a folder name, possibly preceded by a volume and path spec. If *file* is omitted, PRINT prints the text contained in the console window.

Switches

/P

prompts you for confirmation before printing each file.

/D

displays the standard Print dialog before printing the file. When printing a wildcarded series of files, the dialog is only displayed before printing the first file of the series. The parameter settings are then applied to all following prints until you execute PRINT/D once more.

MacDOS vs DOS

Options and parameters

MacDOS supports none of the DOS options, but it can prompt you with the switch **/P** and **/D**. Only one parameter is accepted by MacDOS, but it can be wildcarded or specify a folder.

Notes

Page Setup and Print console menu items

The "Page Setup..." item in the "File" menu behaves in the usual way. The "Print console..." item prints what is in the MacDOS window.

Print dialog box

By default, the command PRINT does not put up the usual Macintosh dialog box. Thefeore, it lets you print several text files without having to click the mouse for each one of them. It normally prints one copy of each entire file but you can change the default via the "Print console..." item of the "File" menu or the switch **/D**. MacDOS will then apply the same settings when executing PRINT until you change them again.

Form Feeds

When MacDOS encounters a line which begins with a FF character (cntl-L = 0x0C), it executes a form feed without printing that line.

Examples

```
print "\myProject folder\sources\*.c"
print \mixedFiles /p
print "a file" /D
```

Frequently occurring errors

`E27: File or directory not found`
The parameter did not identify any file at all or one of the directories specified in the path could not be found.

`E40: Not a file of type 'TEXT'`
You can only `PRINT` text files. Perhaps you forgot to save it from your word processor as "text only".

PROMPT

changes the command prompt.

Syntax

`prompt` [*`text`*]

`"PROMPT text"` is equivalent to `"SET PROMPT=text"`

Parameters

`text`
specifies what you want to have in the command prompt. When `PROMPT` encounters in *`text`* a dollar sign, it interprets the following character as a request to insert special strings into the command prompt. The possible requests are:

`$B`	| (pipe)
`$D`	current date
`$E`	escape (ASCII code 27)
`$G`	> (greater-than sign)
`$H`	backspace (erases the previous character)
`$L`	< (less-than sign)
`$N`	current volume
`$P`	current volume and path
`$Q`	= (equal sign)
`$T`	system time
`$V`	MacDOS version number
`$_`	new-line
`$$`	$ (dollar sign)

When *`text`* is missing, `PROMPT` resets the command prompt to the default string (`$N$G`).

MacDOS vs DOS

The two implementations are identical.

Notes

Date and Time formats
The date and time formats are set with `DATE` and `TIME` respectively.

Paths and dismounted volumes
If you `EJECT` a floppy while you are attached to it and your command prompt contains `$P`, you will get the error message "`No such volume`" after every prompt. To avoid such an error message you should attach to a mounted volume. Naturally, you could also remove the `$P` from the prompt string.

Examples

command	**prompt**
`prompt $p$g`	`C:\MacDOS folder>`
`prompt "Now: $t$$"`	`Now: 13:33:16$`

```
prompt $d;$t$_$v !          94-09-30;13:35:10
                            2.0.0 !
```

See Also
DATE, TIME, VER, VOL, CD

RD, RMDIR

delete an empty folder (ie. ReMove a DIRectory).

Syntax
rd *name*
rmdir *name*

The two commands are functionally identical.

Parameters
name
is the name of the folder to be removed. It can include a volume and path spec.

MacDOS vs DOS
The two implementations are identical.

Notes
RMDIR vs. DEL
DEL followed by a folder name deletes all the files in the folder, while RMDIR followed by the same folder name actually removes the folder (if it is empty).

Examples
Removes a folder from the root directory:
```
rmdir \foldName
```
Removes a subfolder of the current folder:
```
rmdir subFold
```
Removes a folder from the same folder which contains the current one:
```
rmdir ..\sisterFold
```
Removes a folder from the current folder of volume D:
```
rmdir D:oldFold
```
Removes a folder from the root directory of volume D:
```
rmdir D:\aFold
```

Frequently occurring errors
E49: File busy, dir non empty, or path error
RMDIR can only remove a folder if it is empty.

See Also
MD, MKDIR, CD, CHDIR

READ

reads a line of text from a file opened with OPEN.

Syntax
read *fileID* [*var*]

Parameters

fileID
is the number returned by OPEN. The file can be closed with the command CLOSE, but MacDOS closes it automatically when it encounters the EOF.

var
is the name of the global variable into which the read line is to be stored. If *var* is omitted, MacDOS displays the line in the MacDOS window.

MacDOS vs DOS

DOS does not support READ.

Notes

Line termination
READ strips the newline character (CR) at the end of each line containing more than one character. If the CR is the only character in the line, MacDOS replaces it with a space. This guarantees that the line returned is never empty.

Examples

`read 3`	reads a line from file #3 and displays it on the monitor's screen.
`read %fileID% aLine`	reads a line from the file whose ID is stored in the variable `fileID` and stores it into the variable `aLine`.
`read %2`	reads a line from the file whose ID was passed to the batch program via the second replaceable parameter.

Frequently occurring errors

`E3: File not opened by the user`
The file you wanted to READ was not opened. Remember that MacDOS automatically closes the file when you reach the EOF.

See Also

OPEN, WRITE, CLOSE

REN, RENAME

change the name, the creator, and the type of a file or group of files. These commands also assign the current date/time and toggle the 'hidden' file attribute.

Syntax

ren *source* [*destination*] [**/!**] [**/C=***src creator*] [**/C!***dst creator*] [**/D**] [**/H**] [**/L**] [**/T=***src file type*] [**/T!***dst file type*]

The two commands REN and RENAME are functionally identical.

Parameters

source
is a filename preceded by an absolute or relative path which specifies the file or group of files to be renamed. A group of files is identified via a wildcarded filename or a directory name. To rename a folder rather than its contents, refer to RENDIR.

destination
is a filename which specifies the new name[s]. If omitted, it is assumed to be identical to the source argument.

Switches

/!
enables automatic update of destination names in case of duplications. When RENAME finds that a file with the same name already exists, it extends the new filename with an alphanumeric character (0..9, a..z).

/C=*src creator*
/C!*dst creator*
with the equal sign, **/C** specifies that only files of a particular creator are to be renamed. With the exclamation mark, it specifies that the creator of the file[s] is to be changed to the specified value. Note that no spaces are allowed on either side of the equal sign or exclamation mark, and that the creator must be four characters long. Therefore, if a creator includes space characters you must double quote the switch (eg. `ren /c=ABCD "/c=ABC "` would select all files in the current directory created by `'ABCD'`and change their creator to `'ABC '`).

/D
changes the date and time of last update to the current date and time.

/H
toggles the 'hidden' file attribute. A hidden source becomes visible, and a visible source becomes hidden. This switch is only legal when the source argument specifies a single file.

/L
makes the selection of source files case sensitive.

/T=*src file type*
/T!*dst file type*
with an equal sign, **/T** specifies that only files of a particular type (eg. `'TEXT'`) are to be renamed. With the exclamation mark, it specifies that the type of the file[s] is to be changed to the specified value. Note that no spaces are allowed on either side of the equal sign or exclamation mark, and that the file type must be four characters long. Therefore, if a file type includes spaces, you must double quote the switch (eg. `ren /t=ABCD "/t=ABC "` would select all files in the current directory of type`'ABCD'`and change their type to `'ABC '`).

MacDOS vs DOS

destination
DOS returns an error if the destination is identical to the source. MacDOS accepts equal filenames, although it only performs the renaming when necessary. The main reason for relaxing the check is that MacDOS provides the possibility of changing some file attributes without having to change the filenames.

Switches
MacDOS provides several switches while DOS provides none.

Usage of wildcards
MacDOS has a better wildcarding mechanism than DOS (see below for details).

Notes

Upper and lower case
Unless you specify the option **/L**, characters in upper and lower case are considered identical when filtering source files. On the other hand, with or without **/L** the new destination filename will be as typed by you. Note that because the Mac OS does not make distinction between upper and lower case when identifying files, a command which just changes from upper to lower case or vice versa a single filename without actually changing characters returns the error "duplicate filename". You must perform such a change in two steps or use wildcards.

Changing visibility
You are only allowed to change the visibility of one file at a time. That is, you cannot wildcard the filename when you use **/H**.

Wildcarding strategies
MacDOS memorises adjacent wildcard characters as a single wildcard sequence characterised by the minimum and maximum lengths of possible matching strings. The minimum length of a matching string coincides with the number of question marks in the sequence. The maximum length of a matching string also coincides with the number of question marks but only if there are no asterisks. If there are one or more asterisks, the maximum length of a matching string is only limited by the maximum length of filenames (31 characters). Note that because of this strategy, wildcarding strings containing at least one asterisk and having the same number of question marks are equivalent (eg. `"*?*?"`, `"**??"`, `"?*?"`, `"?****?**"`, etc). In fact, there is no reason to use more than one asterisk in a string of adjacent wildcard characters, and the ordering of the question marks and the single asterisk is irrelevant.

When you specify a folder as source parameter, MacDOS renames all the files in that folder.

Normally, each wildcarding section of the destination consists of a single asterisk, as the main function of `RENAME` is to replace fixed parts of source filenames with fixed strings which appear in the destination. That is, MacDOS only uses the wildcards in the destination to match corresponding wildcard sections in the source. Therefore, wildcard sequences in the destination usually have a maximum length equal to or greater than the corresponding sequences in the source. Nevertheless, combinations with a wildcard sequence in the destination shorter than the corresponding one in the source are legal. In that case, the source string is truncated before being inserted into the destination. For example, `"REN *.* ????????.???"` cuts all filenames containing a period to eight characters before and three characters after the period. Names with fewer characters than the maximum length of the wildcarding sequences in the destination are left unchanged. If, on the other hand, the minimum length of a wildcarding sequence in the destination is greater than the number of characters matched by the corresponding sequence in the source, the characters are copied to the destination filename WITHOUT padding or repetitions. In conclusion, `"REN *.txt ???.doc"` renames `a.txt` to `a.doc`, `ab.txt` to `ab.doc`, and `abcd.txt` to `abc.doc` .

The renaming with wildcarding sections in the destination shorter than the corresponding sections in the source is somewhat dangerous, because different filenames might end up with the same name.

Examples

When the source contains AS MANY wildcard sections as the destination, they are matched in sequence. The characters explicitly given in the source are then replaced by the corresponding characters of the destination:

no wildcards:

`REN b.ext blob`	renames `'b.ext'` to be `'blob'`

one wildcard section:

`REN *TXT *DOC`	replaces all terminating `"TXT"`s with `"DOC"`s
`REN ??? *.TXT`	attaches the extension `".TXT"` to all filenames three letters long
`REN aFold bla_*`	`"bla_"` is prepended to all filenames in the folder `'aFold'`
`REN abc* *`	`"abc"` is removed from all filenames which begin with it

two wildcard sections:

`REN *LOW* *low*` replaces with `"low"` all sequences which match `"LOW"` first encountered in each filename (ie. `"Low"`, `"low"`, `"LoW"`, etc.)

`REN *LOW* *low*/L` replaces with `"low"` the `"LOW"` first encountered in each filename

`REN "* ???" *.*` replaces a space with a period in all filenames where the space occurs three characters before the end

`REN *.?* ????????.*` truncates to eight characters before the period all filenames which contain a period followed by at least one character

`REN *AA* *A*` replaces the first pair of `'A'`s found in each filename with a single `'A'`. What happens if there are two files named, say, `XAAAB` and `XAAB`? MacDOS aborts the operation when it attempts to rename `XAAAB`, because `XAAB` already exists.

`REN/! *AA* *A*` as in the previous example, MacDOS replaces the first pair of `'A'`s found in each filename with a single `'A'`. If there are two files named, say, `XAAAB` and `XAAB`, MacDOS renames `XAAAB` to `XAAB0` because `XAAB` already exists (although later `XAAB` is renamed to `XAB`). If `XAAB0` already exists, MacDOS renames `XAAAB` to `XAAB1` (although later `XAAB0` is renamed to `XAB0`). This "forced" renaming continues by changing `1` to `2`, `3`, ..., `9`, `a`, `b`, ..., and `z`.

When the source contains MORE wildcard sections than the destination, the parts masked in the source by the additional wildcarding sections are left unchanged. This is a way of selecting filenames on the basis of several specific strings but then only replace the first one. For example:

`REN SRC*.* DST*` replaces the first three characters of filenames which contain a period somewhere after the initial sequence (equivalent to: `"REN SRC*.* DST*.*"`)

`REN *.?? *.X` inserts an `'X'` after the period of filenames in which the period is followed by exactly two characters (equivalent to: `"REN *.?? *.X*"`)

`REN * Z` prepends a `'Z'` to all filenames (equivalent to: `"REN * Z*"`)

The source is NOT ALLOWED to contain FEWER wildcarding sections than the destination.

The following example illustrates the problems which can occur when truncating filenames:

`REN A*X A?X` cuts down to three characters all filenames which begin with an `'A'`. What happens if two files are named, say, `AB1X` and `AB2X`? `AB1X` is renamed to `ABX`, but then the renaming sequence must be aborted, because `AB2X` should also be renamed to `ABX` and that is not allowed. With the option `/!`, `AB2X` would be renamed to `ABX0`.

Frequently occurring errors

`E27: File or directory not found`
The source parameter did not identify any file at all or one of the folders specified in the path could not be found. Note that creator and file type are case sensitive. Therefore, `/T=text` does not find any file of type `'TEXT'`.

`E58: Duplicate filename(s)`
A file with the destination filename (or one of the filenames, in case of wildcarded renaming) already existed. There are two cases in which this error message can also be displayed when the `/!` switch is ON:

1. The new name is already 31 characters long and MacDOS cannot append the character which would make it unique.
2. MacDOS has already encountered 36 files with the same new name. In this case, MacDOS has run out of possible extension (10 digits and 26 letters).

See Also
`RENDIR`

RENDIR

changes the name of a single directory.

Syntax
rendir *source destination*

Parameters
source
specifies the directory to be renamed. It can be preceded by a volume and path spec.

destination
specifies the new name of the directory. It cannot be preceded by a volume and path spec.

MacDOS vs DOS
DOS does not support `RENDIR`.

Frequently occurring errors
`E11: Not a directory`
You tried to rename a file with `RENDIR`. Use `RENDIR`.

`E27: File or directory not found`
MacDOS did not find the folder to be renamed or one of the folders specified in its path.

See Also
`MKDIR, RMDIR, RENAME`

RESTART

restarts the Macintosh.

Syntax
restart

MacDOS vs DOS
DOS does not support `RESTART`.

See Also
`SHUTDOWN`

SERIAL

Directs MacDOS to use the serial port[s] for Input/Output.

Syntax

serial [*inout* | **OFF**] [**/M**]

Parameters

inout
identifies the ports to be used for serial communication. It consists of a pair of letters (not case sensitive) which specify the ports to be used for input and output respectively. 'A' indicates port A (the modem port), 'B' indicates port B (the printer port) and '@' indicates that no port should be used.

OFF
disables I/O via the serial port[s].

/M
specifies that MacDOS has just to Monitor the characters read from the serial port instead of interpreting them as commands. The default is that MacDOS does accept commands from the serial port.

Without parameters, `SERIAL` reports the current setting.

MacDOS vs DOS

DOS does not support `SERIAL`.

Notes

Port settings
The ports are set to 9600 bps, no parity, 8 data bits, 1 stop bit, and hardware handshaking.

Working in parallel to the standard I/O
The console window and the keyboard remain fully operational while I/O via the serial port[s] is enabled.

Control characters
MacDOS accepts from the serial ports the following control characters:

`cntl-C`	Aborts the current operation.
`cntl-H`	Backspace, deletes the last character typed.
`cntl-I`	Attempts to complete file and folder names.
`cntl-M`	Terminates a command and starts its execution.
`cntl-Z`	EOF, used in conjunction with `MORE` to store text into a file.

Testing a port
The small utility program `SendToPort` provided on the MacDOS floppy (in the folder `serial comms`) sends characters to a serial port and can be used to test a particular setup.

You can also use `SendToPort` to control a copy of MacDOS on a different system: connect the two Macs via a serial port (eg. the modem port), start `SendToPort` on the "master" Mac and direct it to use port A, then start MacDOS on the "slave" Mac and type at the MacDOS prompt the following command: `"serial a@"`. This will direct the "slave" to accept commands via the modem port without replying. You will then be able to control the "slave" remotely by typing on the "master" commands like:

```
dir^m
confirm on^m
```

You will also be able to abort "slave" operations by typing on the "master" the two characters `^C`.

For additional information on SendToPort, please refer to the README file in the same folder.

Examples

SERIAL AA /M
sets both input and output to the modem port. MacDOS then sends to the port all characters displayed in the console window and displays on the console window all characters received from the same port.

SERIAL BA
sets the input to the printer port and the output to the modem port. MacDOS then sends to the modem port all characters displayed in the console window and accepts characters from the printer port as if they had been typed on the keyboard.

SERIAL BA
sets the input to the modem port and disables the output. MacDOS then accepts characters from the modem port as if they had been typed on the keyboard but does not send any character to either serial port.

Frequently occurring errors

MacOS Error –97
The port is in use. Use the other port if you can. AppleTalk could be the culprit.

SET

displays, sets, and removes global variables.

Syntax

set [*var*=[*value*]]

Parameters

var
is the name of the variable to be SET. If you omit *var*, SET displays the list of variables with their values.

value
is the string to be assigned to the variable. When you omit *value*, SET behaves in different ways depending on the type of variable: system variables are re-initialised to their defaults, while user-defined variables are removed altogether.

MacDOS vs DOS

SETting of system variables
When you set the system variables PROMPT, DIRCMD, and PATH to an empty string, MacDOS sets them to their default values, while DOS removes them.

Spaces and variable names
MacDOS ignores the spaces between the variable name and the equal sign, while DOS does not. The result is that DOS creates variables with names which terminate with one or more spaces. For example, DOS responds to the command "set var =3" by setting a variable named "var " rather than simply "var"!

MacDOS also replaces multiple spaces and tabs within a variable name with single spaces. As MacDOS uses the equal sign to delimit the variable name, you must not double quote variable names.

Notes

Spaces after the equal sign

The variable is SET to whatever follows the equal sign, including any space. Therefore, "set var = a " sets a variable named "var" to " a ".

Variable names
The names are not case sensitive and are actually saved by MacDOS in upper case. If you use double quotes in variable names, you run into trouble when you try to do something with those variables, because in all other commands the double quotes are used to delimit names and are not considered to be part of the names themselves. This anomaly of SET was introduced to be consistent with DOS.

Examples

```
set a numeric var =      33
set ZZZ=strings do not need to be enclosed in double quotes
set prompt=
```

See Also

INCR, DECR, TOUPPER, SSTR, PATH, PROMPT, DIR

SHIFT

moves the replaceable parameters of a batch program forward one position.

Syntax

shift

MacDOS vs DOS

The two implementations are identical.

Notes

SHIFT allows you to execute batches with more than 9 user-defined replaceable parameters (the current maximum is 24).

After MacDOS has executed the command SHIFT, the replaceable parameter 0 contains what was previously in 1, 1 contains what was in 2, etc. The last replaceable parameter (ie. 9) contains what you typed as 10th parameter of the batch program and was previously unaccessible.

By using SHIFT repeatedly, you gain access to all the strings you typed after the batch name when you started the program. This is particularly useful if you need to perform the same operation on all parameters. In that case:

1. Write the code for %1.
2. After performing the operation the first time, execute SHIFT and re-execute the code from the beginning. The new %1 will then contain what during the initial pass was in the second parameter.
3. Repeat the same sequence several times until you find that %1 is empty.

Examples

This batch uses SHIFT to process all replaceable parameters, one at a time:

```
:LOOP_LBL
    if "%1x" == x goto END_LBL
    perform on %1 the required task
    ...
    shift
    goto LOOP_LBL
:END_LBL
```

SHOW

retrieves the message corresponding to an error code.

Syntax

show *err* [*var*]

Parameters

err
is the error code

var
is the name of the variable where SHOW is requested to store the message. If *var* is omitted, SHOW displays the message on the screen. If MacDOS does not "know" an error code, SHOW just displays the error code itself.

MacDOS vs DOS

DOS does not support SHOW.

Notes

Additional parameters
SHOW ignores additional parameters.

SHOW, DOSERR, and ONERROR
SHOW is particularly useful to implement your own error handling in a batch program, especially when executing with ECHO OFF. You can trap errors with ONERROR, jump to a fixed command line, and then use the system variable DOSERR to decide what action is necessary. Also, you can use the command SHOW %DOSERR% to display error messages in clear:

```
          ! setup error trapping
          onerror ERR_LBL
          ! setup return label
          set RET =LBL_1
          set DOSERR=0
          ! *** here insert the command for which you want ***
          ! *** to trap the error                           ***
     :LBL_1
          ...
          goto DONE_LBL

     :ERR_LBL      error-handling procedure
          if %DOSERR% == ...
          show %DOSERR%
          ! reset error trapping and resume normal execution
          onerror
          goto %RET%

     :DONE_LBL     come here to terminate the batch program
```

Examples

```
show 27
```
returns the string File or directory not found

```
show 333
```
returns the string 333

```
show %var1% var2 the rest of the line is ignored
```
stores into var2 the error message corresponding to the code stored in var1

```
show abcd efgh ijkl
```
sets the variable efgh to the string abcd

See Also
ONERROR

SHUTDOWN

shuts down the Macintosh.

Syntax
shutdown

MacDOS vs DOS
DOS does not support SHUTDOWN.

See Also
RESTART

SSTR

extracts SubSTRings from variables.

Syntax
sstr *var* [*delim*] [**/E**] [**/L** | **/R**]

Parameters
var
is the name of the variable from which the substring is to be extracted. SSTR replaces the content of *var* with the substring.

delim
is the string which delimits the substring. MacDOS searches *var* and stops when it finds *delim*. Depending on the presence of switches, MacDOS extracts from *var* the substring on the left or on the right of *delim*. If you omit *delim* or *delim* is empty, MacDOS leaves *var* unchanged. MacDOS aborts the command and reports an error when *var* does not contain *delim*.

Switches
/E
searches *var* from the End (ie. from right to left). When **/E** is omitted, the search is done from the beginning (ie. from left to right).

/L | **/R**
specifies whether the substring is to be extracted on the Left or on the Right of *delim*. If you omit both **/L** and **/R**, **/L** is assumed. If you include in the command both **/L** and **/R**, the command is rejected.

MacDOS vs DOS
DOS does not support SSTR.

Notes
Upper and lower case
Searches are always case sensitive. If you need to perform a case-insensitive search, convert first *var* and *delim* to uppercase with TOUPPER.

Examples
```
set var=a few chars and nothing more
sstr var "and nothing"      var now contains: "a few chars "
```

```
sstr var /E " "          var now contains: "a few chars"
sstr var w /R            var now contains: " chars"
sstr var a /L            var now contains: " ch"
```

Frequently occurring errors

`E72: String search failed`
the string *delim* could not be found within *var*.

See Also

SET, INCR, DECR, TOUPPER

SUBSTVOL

creates a literal volume ID.

Syntax

substvol *letter vol*

Parameters

letter
is a character between 'A' and 'Z' (case insensitive). *letter* can then be used in place of *vol* in all commands.

vol
is a volume identifier (1 is the start-up volume). Note that the volume does not need to be mounted.

MacDOS vs DOS

DOS does not support SUBSTVOL.

Notes

SUBSTVOL was introduced to make the porting of batch programs from DOS easier by allowing you to use letters instead of numbers when identifying volumes.

See Also

VOL

TIME

displays and sets the system time and the time format. MacDOS uses the format set through TIME whenever it needs to display a time (starting a LOG file, in a DIR listing, etc.).

Syntax

time [*time*]

Parameters

time
is the time which should become the new system time. Also, the format of the new time becomes the new format used by MacDOS whenever it displays a time. If you type midnight (ie. 00:00:00a for a 12h format and 24:00:00 for a 24h format), MacDOS only updates the time format and leaves the system time unchanged. If you omit *time*, MacDOS displays the current system time in the current format and prompts you for a new time. You can then type the new time, or hit Carriage Return if you do not want to change anything.

MacDOS vs DOS

DOS does not let you change the clock format via the command `TIME`.

Notes

12h vs. 24h clocks

To change between 12h and 24h clock formats, you only need to type a time in the new format. The best way of doing it is to include in autoexec.bat the setting of midnight. By default, time is displayed in 24h format with colons as separators.

Partial time specification

You can omit to specify the seconds or both minutes and seconds. In that case, MacDOS sets the missing fields to zero. Note that a time specification cannot terminate with a separator. Therefore, times like `17:20` and `18` are legal, while `17:20:` and `18:` are not. Also, you cannot omit intermediate fields. Therefore, times like `16::45` and `:20:30` are illegal.

Examples

command	old time	new time
`time 00:00.00`	`16:03:20`	`16:03.20`
`time 00:00.00`	`04:03:20p`	`04:03.20p`
`time 00:00.00a`	`16:03:20`	`04:03.20p`
`time 00:00.00a`	`04:03:20p`	`04:03.20p`
`time 24;00;00`	`16:03:20`	`16;03;20`
`time 24;00;00`	`04:03:20p`	`16;03;20`
`time 17:20/03`	`16:03:20`	`17:20/03`
`time 17:20/03`	`04:03:20p`	`17:20/03`
`time 08-15-30`	`16:03:20`	`08-15-30`
`time 08-15-30a`	`16:03:20`	`08-15-30a`
`time 17`	`16:03:20`	`17:00:00`
`time 17:7`	`16:03:20`	`17:07:00`
`time 17:7`	`04:03:20p`	`17:07:00`

Frequently occurring errors

`E52: Invalid time`

Probably one of the fields was outside its valid range.

See Also

`DATE`

TOUPPER

converts variable contents to uppercase.

Syntax

`toupper` *`var`*

Parameters

`var`

is the name of the variable to be converted. `TOUPPER` replaces the string contained in *`var`* with its uppercase version.

MacDOS vs DOS

DOS does not support `TOUPPER`.

Notes

Diacritical marks

`TOUPPER` converts strings with the standard function provided by the Macintosh Toolbox. Therefore, characters with diacritical marks are converted to uppercase correctly (ü to Ü, ö to Ö, etc.).

See Also

DECR, INCR, SSTR

TREE

lists the contents of a folder by displaying its tree structure graphically.

Syntax

tree [*folder*] [**/**[**–**]**A**[[:]*attribute list*]] [**/**[**–**]**B**] [**/C=***creator*] [**/F**] [**/**[**–**]**L**] [**/**[**–**]**O**[[:]*ordering list*]]
[**/**[**–**]**P**] [**/**[**–**]**S**] [**/T=***file type*] [**/**[**–**]**W**]

Parameters

folder
identifies the folder at the root of the tree. It is a folder name possibly preceded by a path and volume spec. If you omit *folder*, TREE lists the contents of the current folder.

Switches

TREE accepts all the switches of the DIR command, although **/B** and **/W** have no effect. The switch **/F**, accepted for compatibility with DOS, has also no effect.

The options set for DIR through the system variable DIRCMD also apply to TREE.

Please refer to the section on DIR for the details.

MacDOS vs DOS

Format of the output
MacDOS always displays the tree list in ASCII and does not implement the switch **/A** of DOS.

DIR switches
The MacDOS implementation of TREE supports all the switches supported by DIR.

Switches
By default, MacDOS always displays the files. Therefore, the switch **/F** of DOS does not have any effect.

Notes

Listing of files
MacDOS by default lists the files. You can suppress it with the switch **/A:–F**.

Recursive listing
By default, MacDOS only lists the contents of the current folder. To list the contents of subfolders hierarchically, use the switch **/S**.

Examples

Please refer to the section about the command DIR.

See Also

DIR

TYPE

displays the content of a text file.

Syntax
type *file* [**/H** | **/R**]

Parameters
file
is the name of the file to be displayed, possibly preceded by a volume and path spec. Unless you use one of the switches, TYPE only displays files of type 'TEXT'.

Switches
/H
displays the data fork of *file* in HEX and ASCII. Non-printing characters are replaced by periods (ie. dots). **/H** is incompatible with **/R**.

/R
displays the resource fork of *file* in HEX and ASCII. Non-printing characters are replaced by periods (ie. dots). **/R** is incompatible with **/H**.

MacDOS vs DOS
DOS does not support the two switches to produce HEX dumps of non-text files.

Notes
Long files
You can pause the listing by typing a cntl-S and resume it with a cntl-Q.

Examples
```
type autoexec.bat
type c:\myDir\aTextFile
type/h data1 > \myHexDumps\data1.hex
type aFile.rsrc /r | more
```

Frequently occurring errors
E40: Not a file of type 'TEXT'
Without switches, TYPE can only display text files.

See Also
MORE

VER

displays the current version of MacDOS.

Syntax
ver

MacDOS vs DOS
The two implementations are identical.

VERIFY

directs MacDOS to verify that files are written correctly when copied.

Syntax
verify [ON | OFF]

Parameters

`[ ON | OFF ]`
`VERIFY ON` directs MacDOS to verify the correctness of disk-write operations when copying. `VERIFY OFF` disables the verification function. Without parameters, `VERIFY` reports the current setting.

MacDOS vs DOS

The two implementations are identical.

Examples

```
verify ON
VERIFY
verify off
```

VOL

displays the volume name.

Syntax

vol [*volume*]

Parameters

volume
is the volume ID. If you omit *volume*, `VOL` displays the name of the current volume.

MacDOS vs DOS

Serial number
MacDOS does not display the serial number of the volume.

Write-protection
MacDOS notifies you if the volume is write-protected.

See Also

`CD, SUBSTVOL`

WRITE

writes a line of text to a file opened with `OPEN`.

Syntax

write *fileID what*

Parameters

fileID
is the number returned by `OPEN`. You can close the file with the command `CLOSE`.

what
is the line of text to be appended to the file. Note that the text must be enclosed in double quotes if it contains spaces.

MacDOS vs DOS

DOS does not support `WRITE`.

Notes

Line termination

WRITE appends a Carriage Return character (CR) to each line of text before storing it into the file.

Examples

```
write 3 "You bet!"
```
adds to file #3 a line of text containing the string: "You bet!".

```
write %fileID% %aLine%
```
adds the text contained in the variable aLine to the file whose ID is stored in the variable fileID.

Frequently occurring errors

```
E3: File not opened by the user
```
You attempted to WRITE a file without OPENing it.

See Also

OPEN, READ, CLOSE

XCOPY

copies folder contents and whole hierarchies of folders.

Syntax

xcopy *source* [*destination*] [**/C=***creator*] [**/D:***date*] [**/E**] [**/P**] [**/S**] [**/T=***file type*] [**/V**]

Parameters

source
is the name of the folder to be copied, possibly preceded by a path and volume spec. Hidden files are not copied.

destination
is the name of an existing folder into which *source* is to be copied, possibly preceded by a path and volume spec. It defaults to the current folder. As files and folders cannot be copied onto themselves, *source* and *destination* must specify different folders.

Switches

/C=*creator*
specifies that only files of a particular creator are to be copied. Note that no spaces are allowed on either side of the equal sign and that the creator string is four characters long. Therefore, if a creator includes spaces, you must double quote the switch (eg. xcopy \ "/c=ABC " copies all the files created by 'ABC ' in the root folder). This switch does not apply to folders.

/D:*date*
only copies files changed on or after the specified date.

/E
recursively copies all [sub]folders, even if empty.

/P
prompts you for confirmation before creating each destination file.

/S
recursively copies subfolders, except if empty.

/T=*file type*

specifies that only files of a particular type (eg. `TEXT`) are to be copied. Note that no spaces are allowed on either side of the equal sign and that the file type is four characters long. Therefore, if a file type includes spaces, you must double quote the switch (eg. `xcopy \ "/t=ABC "` copies all the files of type `'ABC '` in the root folder). This switch does not apply to folders.

/V
re-reads destination files to verify that they have been copied correctly.

MacDOS vs DOS

Switches
MacDOS does not implement **/A**, **/M**, and **/W**, but introduces the new switches **/C** and **/T**. MacDOS assumes **/S** when it encounters **/E**, while with DOS the two switches are independent.

Destination is a folder
MacDOS only accepts a folder specification as destination.

Folders replacing files
When a folder in *source* has the same name as a file in *destination*, MacDOS deletes the file and then creates the folder, thereby effectively replacing a file with a folder. Nevertheless, the item identified in the parameter *destination* itself must be a folder.

Folders becoming empty
With the switch **/S** (not **/E**), only non-empty folders are copied. Nevertheless, this can result in empty folders in the destination tree when **/C** or **/T** filters are applied and no files in a particular source folder passes them.

Notes

Aliases
Aliases are not resolved. Therefore, aliased files and folders appear in the destination folder as they appeared in the source (ie. as aliases).

Specifying dates
When you use the switch **/D** and one or both of the date field separators are slashes, enclose the switch in double quotes. This is necessary to prevent MacDOS from interpreting a date field as a switch.

Examples

`xcopy dir1 dir2`
copies the contents of the folder `dir1` into the folder `dir2`. Both `dir1` and `dir2` are subfolders of the current folder.

`xcopy/s dir1 a: "/d:93/09/20"`
copies the contents of the folder `dir1` to drive `A`. It reproduces the hierarchical structure of `dir1` but only copies files modified on or after September 20th, 1993.

`xcopy/e dir1 dir1Struct /c=mDOS /t=ABCD`
duplicates the hierarchical structure of `dir1` in the folder `dir1Struct`. All destination folders remain empty because MacDOS (creator `'mDOS'`) never generates files of type `'ABCD'`. Any non-existing creator, file-type, or combination of them will achieve the same result.

Frequently occurring errors

`E27: File or directory not found`
The source parameter did not identify any file at all or one of the folders specified in the path could not be found. Note that creator and file type are case sensitive. Therefore, `/T=text` will not find any file of type `'TEXT'`.

`E28: Bad switch`
Perhaps you failed to specify a creator or file type correctly. The equal sign is mandatory and all four characters which form the `OSType` must be provided.

See Also

`COPY`

Appendix A

Error Codes

A Error Codes

This section provides a full list of all error and warning messages displayed by MacDOS. The messages are listed in alphabetical order and preceded by the corresponding error or warning code.

Note that the error codes are only visible when errors and warnings are reported with alert boxes (ie. after executing `ALARM ON`).

If you detect a bug or an undocumented error, please inform Rainbow Hill immediately.

Errors

Attempted to open too many files (E32)

MacDOS attempted to open more files that the Mac OS allows. This is usually due to a high "depth" of batch `CALL`s combined with file copying, output redirection, and logging.
Suggested actions:

- Reduce the maximum "depth" of batch `CALL`s by serialising calls whenever possible. If you are copying files within the "deepest" batch program, split the copying of data and resource forks into two separate commands.

Bad command or file name (E41)

MacDOS could not make sense of your command.
You probably mistyped a command, or attempted to execute a program which MacDOS could not find.
Perhaps you typed a "pipe" at the end of a command, attempted to run interactively a command which is only legal in batch mode, or typed a badly formatted command.
Suggested actions:

- Check that the command name is correct.
- If you were trying to run a program, check whether the batch or application file exists. Also type the command `PATH` and check that it lists all the folders that you expect to find there.
- If you typed a command, check that it is legal in interactive mode.
- Check that you have not omitted any compulsory token in complex commands like `IF`, `DECR`, etc.
- If you started the command with a MacDOS filter, check that you redirected its input.

Bad switch (E28)

You entered a command with a badly formatted switch.
Suggested actions:

- Use the on-line help to look at the switches supported by the command which failed.

Command not implemented (E43)

You attempted to use a DOS command which is not currently supported by MacDOS.
Suggested actions:

- Protest with Rainbow Hill. If we received enough protests, we might implement it in a future release.

Command only executable in batch (E46)

You typed a command which is only executable within a batch program. Currently, only `GOTO`, `IF`, `NEXT`, `PAUSE`, and `SHIFT` return this error.

Suggested actions:

- Re-examine what you intended to do and perhaps write a small batch program.

Dimensioning: prompt too long (E15)

You attempted to set a prompt longer than 199 characters. Do you really want to be prompted by a couple of lines of text after typing each command?

Suggested actions:

- Be sensible!

Dimensioning: too many arguments (E6)

You entered a command with 24 or more parameters and switches.

Suggested actions:

- Check whether some switches appear more than once and get rid of the duplicates. If there are no duplicates, could you please tell Rainbow Hill what command you typed?

Dimensioning: too many commands (E5)

MacDOS counted more than 8 commands separated by pipes in a single line.

Suggested actions:

- Break up the chain of piped filters by storing intermediate results into disk files.

Dimensioning: too many tokens (E2)

Your command line contained 64 tokens or more.

Suggested actions:

- This situation is very unlikely to occur, because MacDOS limits the input to a length of 199 characters. Please, inform Rainbow Hill of the command which caused the problem.

Directory name missing (E23)

You entered without parameters a command which expected a folder name as a parameter.

Suggested actions:

- Re-examine what you were trying to do. You probably typed `MKDIR \` or `RMDIR \`. If that is the case, remember that you cannot ever remove a root directory.

Directory specified (E53)

MacDOS found a directory when it expected to find a file.

Suggested actions:

- If you were executing an `XCOPY` command, you will find that a file in the source and a folder in the destination have the same name. `XCOPY` then returns this error when it attempts to replace the folder in the destination. Rename the file or remove the folder.

Disk full (E80)

MacDOS could not create or extend a file because there is not enough space on the requested volume. Note that when MacDOS needs to overwrite a file during a`COPY` operation, it first creates a new file with a temporary name.

Suggested actions:

- Make some space by removing existing files.

DOS command not implemented (E1)

You attempted to use a DOS command which is not supported by MacDOS.

Suggested actions:

- Protest with Rainbow Hill, although the chances that you are going to get this command in a future release are slim.

Double initialisation of pipes (E90, pipeDoubleInitErr)

MacDOS attempted to initialise twice the pipe mechanism. This might be due to MacDOS' memory partition being corrupted.

Suggested actions:

- This situation should never occur. Start logging with `LOG/O`, attempt to reproduce the problem, and, if successful, send a listing of the log file to Rainbow Hill.
- Quit and re-launch MacDOS.

Duplicate file name and version (E48)

Up to release 0.2.0, MacDOS returned this error whenever a File Manager call returned the Mac OS error -48 (dupFNErr). This occurs whenever one tries to rename a file or create a directory and an item with the same name in the same folder already exists. After 0.2.0 this errors have all been trapped with **E58 Duplicate filename(s)**.

Suggested actions:

- Upgrade to the latest release or refer to **E58**.

Duplicate filename(s) (E58)

You attempted to create a directory or rename a file when an item with the same name and in the same folder already existed.

Suggested actions:

- Either you use a different name, or you rename the existing item, or you operate in a different folder.

Error when following a path (E17)

MacDOS could not follow a chain of folder names given in a command parameter.

Suggested actions:

- Check that you did not attempt to "back up" higher than the root directory of a particular volume. This happens when you type too many up-one-level-strings (ie. "..\") in a path.

File already open for writing (E77)

You attempted to `OPEN` a file already opened by another application in exclusive mode.

Suggested actions:

- Check that you correctly typed filename [and path].
- Check that you are not `LOG`ging into that file and remember that only files opened with `OPEN` are listed when you type `OPEN` without parameters (ie. the file used for logging is opened internally by MacDOS and does not appear in the list).
- If you are convinced that the file should be closed, quit the application which opened it last. That application might have erroneously left the file open (a bug). If in doubt, close all other applications before attempting to open the file with MacDOS.

File busy, dir non empty, or path error (E49)

MacDOS attempted to access or remove an open file or to remove a folder which was not empty.

Suggested actions:

- Check that no other application is using the file that you were trying to access. If this does not work, close all other applications, because some of them might have a bug and hold onto files.
- If you got this error message while trying to remove a folder with `RMDIR`, empty it by deleting its content and then try to remove it once more. Note that the folder might contain an invisible file, like those used to show folders with custom icons.
- If your command contained a path, check that it is correct: you might have been accessing an item with the correct name but in the wrong folder.

File not opened by the user (E3)

You attempted to access with `READ` or `WRITE` a file which you did not open with `OPEN`. Remember that MacDOS closes the file when you read past the EOF.

Suggested actions:

- Check that you typed the correct fileID and type `OPEN` without parameters to see whether some other files are open.
- Open the file before attempting to access it again.

File open for writing (E75)

You attempted to `READ` from a file `OPEN`ed for writing.

Suggested actions:

- Check that you typed the correct fileID and type `OPEN` without parameters to see whether some other files are open.

File open for reading (E74)

You attempted to `WRITE` into a file `OPEN`ed for reading.

Suggested actions:

- Check that you typed the correct fileID and type `OPEN` without parameters to see whether some other files are open.

File or directory not found (E27)

MacDOS could not find the item you wanted to operate on.

Suggested actions:

- If your command contained a path, check that it specifies the correct folder: you might have been looking for something which exists somewhere else.
- If you were specifying a wildcarded name, this error could simply mean that no name matched the wildcarding.

Filter error (E70, pipeFilteringErr)

During execution of a command containing pipes, one of the filters reported to MacDOS a generic error.

Suggested actions:

- If you are developing a filter, always try to report specific errors or at least associate a proprietary error message to the generic error code. Please refer to Appendix B for help.

Filter sent a command that only MacDOS should issue (E92, pipeComFilterE)

While performing a command containing pipes, MacDOS received from a filter a message of a type which is normally used by MacDOS itself to configure a chain of filters.

Suggested actions:

- You have probably used the wrong command code while modifying pipe.c or pipe.h. You should not modify those files. Please refer to Appendix B for help.

Filter sent a data message out of a sequence (E93, pipeSequenceE)

While performing a command containing pipes, MacDOS received a data message with an invalid sequence number.

Suggested actions:

- You have probably dropped a message in your filter. Remember that you have to forward to the outgoing pipe a message for each message that you receive from the incoming pipe.
- If you were modifying pipe.c or pipe.h, you have probably corrupted the variable used for storing the last sequence number. You should not modify those files. Please refer to Appendix B for help.

Filter sent downstream a message that should only be sent upstream (E101, pipeDownstreamErr)

While performing a command containing pipes, MacDOS received an error message through the PPC port that should only deliver data messages.

Suggested actions:

- You have probably caused the problem by incorrectly updating pipe.c or pipe.h. You should not modify those files. Please refer to Appendix B for help.

Filter sent upstream a message that should only be sent downstream (E100, pipeUpstreamErr)

While performing a command containing pipes, MacDOS received a data message through the PPC port that should only deliver error messages.

Suggested actions:

- You have probably caused the problem by incorrectly updating pipe.c or pipe.h. You should not modify those files. Please refer to Appendix B for help.

Folders named Desktop Folder and Trash cannot be created or removed (E73)

These two names are reserved.

Suggested actions:

- Slightly change the name so that it becomes acceptable. Something like `Desktop-Folder` and `Trash-Can` would be ok. Remember that file and folder names are case insensitive. Therefore, changing to lower case would not help.

The command cannot be piped (E79)

You tried to pipe the output of something that does not allow output redirection (eg. an application or an AppleScript). You can pipe the output of a standard `FOR`, but only to the command `MORE`.

Suggested actions:

- You would probably achieve your purpose by redirecting the output to a file. You could then `READ` and operate on one line of text at a time within a batch program.
- Check that you have not used a normal application in place of a MacDOS filter.

Hardware volume lock (E51)

MacDOS could not operate on an item because the volume had the "write-protect" switch ON.

Suggested actions:

- Check that you are operating on the correct volume and be sure that you know what you are doing. Usually, write protection has a reason to exist. If you really want to do it, eject the cartridge, remove its write protection, and re-insert it into the drive before repeating the command.

I/O error in disk operation (E56)

MacDOS could not complete a write or verify operation when copying or verifying a file. The number of bytes written or read back was lower than the number requested. The destination file was left corrupted.

Suggested actions:

- Attempt to save critical files to a different volume before doing anything else.
- Reformat or run a diagnostic on the destination volume before using it again.

Illegal wildcarding (E34)

MacDOS was given a wildcarded filename when it expected to operate on an individual file.

The most likely cause is that you specified a folder name as a parameter of a command which expected a file name. Then MacDOS attempted to operate on all files in the folder.

Another possible explanation is that you passed a wildcarded filename to `XCOPY` (which only accepts folder names).

Suggested actions:

- Review what your intentions were and check whether you passed the correct type of parameters to the command you used.

Internal I/O error (E33)

MacDOS encountered an inconsistency when using internally defined global parameters. For example, it might have attempted to use an invalid file ID or a non-existing batch label.

This might be due to MacDOS' memory partition being corrupted.

Suggested actions:

- This situation should never occur. Start logging with `LOG/O`, attempt to reproduce the problem, and, if successful, send a listing of the log file to Rainbow Hill.
- Quit and re-launch MacDOS.

Internal: unknown com state (E10)
Internal: unknown token (E25)

MacDOS encountered an unknown state or token while parsing a command. This is likely to be a case of corrupted memory.

Suggested actions:

- This situation should never occur. Please, inform Rainbow Hill of the command which caused the problem.

Invalid date or date format (E47)

Either you attempted to set an impossible date or date format, or a new date did not conform to the current date format.

Suggested actions:

- Check that year, month, and day were in the order specified by the current date format (eg. 03/25/96 exists in the USA but not in Australia). Also check that the separators were correct.
- Check that the specified day exists (eg. 94/11/31 or 95/02/29 do not exist).
- check that the year is in the interval 1904 to 2040.
- If you were entering a new date format, check that you only used the characters Y, M, and D plus two special characters as separators.

Invalid prompt switch (E14)

MacDOS did not recognise all format specifications included in your `PROMPT` command.

Suggested actions:

- Ensure that each character immediately following a dollar sign is recognised by DOS as a format specification. Remember that you have to type two consecutive dollars if you want to include a '$' in your prompt.

Invalid session reference number (E85, noSessionErr)

During execution of a command containing pipes, one of the filters attempted to address a PPC session that was not initialised.

Suggested actions:

- You have probably caused the problem by incorrectly updating pipe.c or pipe.h. You should not modify those files. Please refer to Appendix B for help.

Invalid time (E52)

The new time given to MacDOS was wrong.

Suggested actions:

- Check that the specified time exists (eg. 13:20p, or 25:00 do not exist).

Label not found (E44)

While running in batch, MacDOS could not complete a command containing a label because it did not find the label in the current batch program.

Suggested actions:
- If the command that failed was a `REPEAT`, replace it with a `GOTO`.
- If the command that failed was a `GOTO`, ensure that the destination label actually exists in the same batch file.
- If you were executing a batch without echo, comment out the `ECHO OFF` command in the batch file in order to determine which command is causing the error. With `ALARM ON`, MacDOS actually tells you the name of the label which could not be found.

Label used for more than one loop (E59)

MacDOS found that two `REPEAT` commands contained the same label name.
Suggested actions:
- Replace the `REPEAT`s with `GOTO`s.

Labels are only allowed in batch mode (E96)

You entered a label interactively.
Suggested actions:
- Just don't do it. In any case, what is the purpose of typing a label if you cannot use a `GOTO` to jump to it?

Labels must not begin with a percent sign (E65)

MacDOS found a label which begins with a percent sign.
Suggested actions:
- Change the offending label name.

Licence number and/or user identification corrupted.
Please reload MacDOS from the original disk. (E67)

MacDOS could not decode the string which normally contains licence number and user name.
Suggested actions:
- Overwrite the faulty copy of MacDOS with that contained in the original disk.

Line parsing error (E9)

MacDOS could not parse the command line. Perhaps, you typed some non-printing characters.
Suggested actions:
- Retype the command (ie. do not recall it with up-arrow). If the problem persists, please report it to Rainbow Hill.

MacDOS error ... (the dots indicate a number)

An error message which provides a code instead of a message in clear should never occur. This indicates that the TEXT resource containing all error messages does not contain a message corresponding to the given code.
Suggested actions:
- Unless you have modified the TEXT resources of MacDOS, please report the problem to Rainbow Hill.

MacDOS requires System 6.0.4 or greater (E38)

MacDOS discovered that the system you are using on your Mac does not contain the Gestalt facility. This facility is necessary in order to properly configure MacDOS.
Suggested actions:
- Upgrade to the latest system 6 (ie. 6.0.8) or to system 7.

MacOS error (E24)

A system trap returned to MacDOS an error code for which MacDOS does not have a message in clear.

Suggested actions:

- If you liked MacDOS to be able to display a clear message rather than a mysterious number, please send the code to Rainbow Hill. Perhaps the next release of MacDOS will then respond to that error condition with a message in clear. If you decide to do so, please send together with the error code a description of the circumstances under which the error was reported.

Max length of filename exceeded (E26)

You specified a file or folder name longer than 31 characters.

Suggested actions:

- Use a shorter name.

Maximum line length exceeded (E76)

MacDOS detected a line of text longer than 199 characters. In batch, this might have happened after variable names where replaced with their content.

Suggested actions:

- Create new variables which only contain what is strictly necessary for the task at hand.
- If your variables contain folder names, use CD to move down the path and use relative paths.

Maximum number of directories exceeded (E20)

MacDOS attempted to access a folder too deep. Only 31 levels of folders are supported by MacDOS.

Suggested actions:

- Attach to an intermediate folder, so that MacDOS does not need to go so deep.
- Flatten the folder tree structure by moving one of the deepest folders up one level. For example, if folder A contains B which contains C, move C out of B so that both B and C are contained in A.

Maximum number of volumes exceeded (E18)

You attempted to handle more than 31 volumes simultaneously.

Suggested actions:

- Do you really need to have simultaneous access to more than 31 volumes? You can probably `EJECT` some of the volumes (which is equivalent to trashing the volume icons).

Memory mode is 32-bit, but application is not 32-bit clean (E31)

Suggested actions:

- Change the memory mode or, better, remove the application from your system. Perhaps a new release of the same application is 32-bit clean.

Mismatch when verifying data written to disk (E57)

MacDOS found that the data read back in order to verify a copy operation did not match what was originally written.

The destination file of the copy operation was corrupted.

Suggested actions:

- Attempt to save critical files to a different volume before doing anything else.
- Re-format or run a diagnostic on the destination volume before using it again.

Nested FORs are not allowed (E62)

You attempted to execute a standard `FOR` within another standard `FOR`.

Suggested actions:

- Replace the outermost `FOR` with a multi-line `FOR`. You could also try to replace it with a loop.

No help available (E64)

MacDOS did not have a help entry for what you required.

Suggested actions:

- Type "`HELP ?`" to obtain the list of the help entries available.
- When asking for help about a MacDOS filter, check that the name is correct.

No such volume (E50)

MacDOS did not find the volume that you specified in one of your commands. Perhaps you trashed a volume but thought that you were only ejecting it.

Suggested actions:

- Check whether you should re-mount a volume which was erroneously dismounted.

Not a directory (E11)

MacDOS found a file when it expected to find a folder. You must have passed to a command a filename instead of the name of a folder. For example, you might have attempted to delete a file with `RMDIR`, to copy it with `XCOPY`, or to rename it with `RENDIR`.

Suggested actions:

- Check the on-line help for the command which failed. You can delete a file with `DEL`, copy it with `COPY`, and rename it with `RENAME`.

Not a file of type `'TEXT'` (E40)

MacDOS found a non-text file when it expected to find a text file. You probably passed the name of a data or application file to commands like `TYPE` or `MORE`.

Suggested actions:

- If you want to display the content of a binary file, use the command `TYPE` with the switch `/H` or `/R`.

Not a licence number (E68)

MacDOS expected to be given a licence number but received a string which did not have the correct format. MacDOS then runs in demonstration mode. This error message has become obsolete with MacDOS 2.0. Therefore, it should have not happened.

Suggested actions:

- Remove the old versions of MacDOS from your system, so that you always execute the latest version.

Not enough memory (E69)

Your Macintosh does not have enough memory to support a MacDOS memory partition.

Suggested actions:

- Click once on the icon of MacDOS and type cmd-I to obtain the Info-box of MacDOS. This will tell you how much memory is necessary to run MacDOS.
- If you are running system 6 without multifinder, the only solution is to add more memory to your Mac.
- Quit one or more applications running under multifinder or system 7, so that enough memory becomes available. As the Mac OS does not rearrange memory partitions, you should first quit the application that you launched last and then progressively quit older applications.

Not enough space in the heap (E16)

MacDOS could not allocate dynamic memory to store temporary data. Commands which need more memory than others are `COPY`, `DIR`, and `FOR`. MacDOS also uses dynamic memory to execute batch programs and store global variables.

Suggested actions:

- When copying wildcarded files with `COPY`, try to break down the copying into two or more separate commands (eg. `A*` could be split into `A*1*` and `A*2*`). This will reduce the number of filenames temporarily stored during the copy.

- If you were CALLing batch programs from within other batch programs, try to reduce the 'depth' of the calls. This will reduce the number of batch files simultaneously open and for which MacDOS needs to keep data in memory.
- If you were executing a FOR, try to tighten up the wildcarding or to break it into more than one loop. This will reduce the number of filenames temporarily stored in each loop.
- It might be useful to get rid of global variables which have been left behind by previous batch programs.
- Increase the size of the MacDOS partition by changing the Preferred and/or Minimum Size shown in the Finder Info-box.

Path with two consecutive backslashes (E21)

Suggested actions:

- Check the path.

Port does not exist at destination (E84, destPortErr)

During execution of a command containing pipes, either MacDOS or one of the filters did not succed in sending a message to a filter.

Suggested actions:

- While developing a filter, you have probably failed to initialise the pipes. Please refer to Appendix B for help.

PPC Port already open. Restart the Mac if you need piping. (E95, portNameExistsErr)

During execution of a command containing pipes, one of the filters did not succeed in opening its PPC port.

Suggested actions:

- If you were developing a filter, you have probably exited the application without going through the appropriate procedure declared in pipe.h. Please refer to Appendix B for help. Unless you want to change the name of your filter, you will have to restart the Mac.

Remote process does not respond (E97, pipeTimeoutE)

During execution of a command containing pipes, MacDOS timed out while waiting for a message from a filter. Also refer to Appendix B.

Suggested actions:

- Perhaps one of the filters is taking longer than expected to process and forward a message. Increase the timeout in seconds by setting the global variable TIMEOUT.
- If you are developing a filter, perhaps you have erroneously dropped a message. Remember that you have to forward to the outgoing pipe a message for each message that you receive from the incoming pipe.
- If you are debugging a filter, perhaps you have prevented it from replying by setting a breakpoint. Increase the timeout by setting the global variable TIMEOUT to a high number of seconds.

Requested variable has not been defined (E63)

MacDOS could not find the global variable you used in a command.

Suggested actions:

- Type SET without parameters to see the list of the global variables currently defined.
- Perhaps you forgot to set the variable before executing the command or mistyped its name in the command.
- Remember that user-defined global variables are totally removed from the system if you set them to nothing (eg. SET VNAME= with no value after the equal sign).

Response addressed to a filter (E94, pipeFilterRespErr)

During execution of a command containing pipes, one of the filters received a response message with its address as destination.

Suggested actions:

- You have probably caused the problem by incorrectly updating pipe.c or pipe.h. You should not modify those files. Please refer to Appendix B for help.

Session was closed (E86, sessClosedErr)

During execution of a command containing pipes, either MacDOS or one of the filters did not succeed in sending a message to a filter.

Suggested actions:

- The destination filter probably failed and exited without reporting an error back to MacDOS. Please refer to Appendix B for help.

Sorry, the operation you have requested needs System 7.0 or greater (E82)

You attempted to use `MEM`, `RESTART`, or `SHUTDOWN` under System 6.

Suggested actions:

- What about upgrading to System 7 ?

Source and destination must be different (E71)

You attempted to `XCOPY` a folder onto itself.

Suggested actions:

- Check where you are. MacDOS assumes the current folder as default if you do not specify a path. You might have also swapped source and destination (the format is "`XCOPY from to`", not "`XCOPY to from`").

Standard FOR cannot execute an IF command (E78)

You attempted to execute an `IF` command after the `DO` keyword of a single-line `FOR`.

Suggested actions:

- Replace in your batch program the standard `FOR` with a multi-line `FOR` as follows:

```
for %var in (theSet) do if ...%var...
```

  becomes:

```
for %var in (theSet) do begin
  if ...%var%...
  next var
set var=
```

- If you were executing the `FOR` interactively, replace it with a short batch program

String search failed (E72)

MacDOS did not find the requested substring in a string variable.

Suggested actions:

- Check the variable name and the delimiter.
- If the error was reported when executing `SSTR` and you are convinced that `it` should have succeeded, type `SET` without parameters to obtain a list of all the variables and their values.
- If the error was reported when executing `DECR`, check that you typed the `BY`-string in the correct upper and lower cases.

Syntax: double input redirection (E7)

You can only redirect the input once. Remember that a command which follows a pipe has its input redirected. An explicit input redirection would cause this error message to be issued.

Suggested actions:

- Recall the command with up-arrow and delete one of the redirections.

Syntax: double output redirection (E8)

You can only redirect the output once. Remember that a command which precedes a pipe has its output redirected. An explicit output redirection would cause this error message to be issued.

Suggested actions:

- Recall the command with up-arrow and delete one of the redirections.

Syntax: invalid arguments (E13)

Not enough or too many parameters and switches for the command you executed. Also, you might have passed the wrong type of parameters (eg. a non-numeric fileID to `READ`).

Suggested actions:

- Check with the on-line help what combinations of parameters and switches are legal.

Syntax: invalid format (E12)

You typed a badly formatted command.

Suggested actions:

- Check that your command did not start with a redirection.
- Check that each redirection token (ie. '<', '>', '>>', and '|') is immediately followed by the appropriate string.

The character requested for quoting is not free (E81)

You attempted to set the global variable `QUOTE` to a reserved character. Reserved characters are: all non printing characters, the space character, the exclamation mark ('!'), the slash ('/'), and the three characters used for redirection and piping ('<', '>', and '|').

The file is not executable (E22)

MacDOS can launch applications and AppleScripts or execute text files as batch programs. You directed MacDOS to execute a file which was of the wrong type.

Suggested actions:

- Use the command `DIR` to list the content of the folder which should contain the executable file; then check the filename.
- If the executable file was supposed to be in a folder identified through the system variable `PATH`, use the command `ECHO %PATH%` to display the current path and check that the folder is correctly specified.
- If you wanted to execute a batch program named `"whatever.BAT"` and only typed after the prompt `'whatever'`, check that there is no file named `'whatever'` in the same folder. If that is the case, you must type the full name of the batch file (ie. `"whatever.BAT"`) in order to execute it.

The pasting of newline characters into the command string is not allowed (E39)

MacDOS does not allow you to paste into the command line more than one line of text. In some cases, you can also get this error message if you hit the return key after MacDOS has displayed a warning.

The path contains too many characters (E83)

While executing `MEM`, MacDOS found that the path of one of the application files was too long to be displayed.

The startup volume cannot be unmounted (E61)

You attempted to unmount with the command `EJECT` the volume which contains the running OS.

Suggested actions:

- Use the command `EJECT/E` to put the startup disk off line without unmounting it.

This Mac does not support PPC (E98, pipeNoPPCErr)

Your Mac does not support the Process to Process Communication toolbox (PPC). Therefore, you cannot use pipes in your commands except to drive `MORE`.

Too many characters in command line (E37)

MacDOS counted more than 199 characters in a line of commands.

Suggested actions:

- Try to break the line into two separate commands:
  - If the line contains the command `MORE`, use an intermediate file and separate output and input redirection instead of a pipe.
  - If you have a long list of files to copy, split the list of filenames.
  - If the length is due to a couple of long paths, use `CHDIR` to move to an intermediate folder where the total length of the paths become shorter (note that a "directory-up" in a path only requires two dots, while a "directory-down" requires the full name of the folder).

Too many labels (E45)

MacDOS found more than 64 labels in a single batch file. MacDOS automatically creates a label for each `FOR–NEXT` pair. Therefore, when counting the labels in a batch file you should also count the multi-line `FOR`s.

Suggested actions:

- Remove from the batch program all the labels which are not used.
- Do not use a multi-line `FOR` when a standard single-line `FOR` can do.
- Break the batch file into two separate files by writing the name of the second file as last command of the first one. Pay attention not to split `FOR–NEXT` pairs and keep all labels together with the corresponding `GOTO`s.
- Extract parts of the batch file which are self-contained and form other batch files which can be `CALL`ed by the main program.

Too many levels of batch CALLs (E30)

The number of `CALL` levels cannot exceed 16.

Suggested actions:

- Check whether you can merge a called batch file into the calling one.
- If a `CALL` is the last command in a batch file, just replace it with the called filename.

Too many sorting requests (E29)

The number of sort options in a `DIR` command cannot exceed 16.

Suggested actions:

- You must have duplicated options, because the total number of options supported is less than 16. MacDOS executes the options in the order in which they are given, from left to right. Therefore, only the rightmost option of each type performs useful work. Get rid of the others.
- If the rightmost sorting option is a sorting by name, you can remove all the preceding options. They are totally irrelevant, because filenames are unique and a sorting by name certainly overrides all previous sorts.

Unexpected data message from a filter (E99, pipeSurpriseDataE)

While performing a command containing pipes, MacDOS received a data message from a filter before it could complete the configuration of the filter chain.

Suggested actions:

- You have probably caused the problem by incorrectly updating pipe.c or pipe.h. You should not modify those files. Please refer to Appendix B for help.

Unknown pipe command (E89, pipeUnknownMessErr)

During execution of a command containing pipes, either MacDOS or one of the filters received a message of an unknown type.

Suggested actions:

- You have probably used the wrong command code while modifying pipe.c or pipe.h. You should not modify those files. Please refer to Appendix B for help.

User Cancelled (E36)

You aborted a piped command with a cntl-C or cmd-Period.

Variables used in iterations must be unique (E66)

MacDOS found in the same batch program two `REPEAT` commands referring to the same label.

Suggested actions:

- Replace the `REPEAT`s with `GOTO`s.

Volume literal ID not assigned (E60)

MacDOS did not find a literal volume ID in its list of definitions.

Suggested actions:

- Type `SUBSTVOL` without parameters to have a list of all literal IDs which have been assigned.
- Change the command to use the correct ID or use `SUBSTVOL` to assign the missing ID to the appropriate volume.

Volume offline (E4)

MacDOS could not complete an operation because the volume specified in your command was off-line. You probably have typed cmd-dot when the Mac OS asked you to insert a particular floppy.

Suggested actions:

- Insert the requested cartridge to mount the volume or use another volume.

Wrong destination in pipe message (E87, pipeAnotherDestE)

While performing a command containing pipes, MacDOS received from a filter a message not directed to MacDOS.

Suggested actions:

- While developing a filter, you must have send a message to a filter ID which does not exist. Please refer to Appendix B for help.

Wrong volume specification (E19)

MacDOS detected an invalid volume ID. Valid volumes IDs are numbers between 1 and 31 and single letters between A and Z (case insensitive).

Suggested actions:

- Replace the volume ID in your command with a valid ID.

(Data message correctly received) (E91, pipeReturnE)
(Do not report this) (E42, pipeDoNotReportE)
(file not found) (E35)
(PPC Operation not completed) (E88)
(wrong file creator) (E54)
(wrong file type) (E55)

These error messages refer to error codes which are used by MacDOS internally. Therefore, you should never see them.

Suggested actions:

- Please report this occurrence to Rainbow Hill. Try to provide as much information as possible about the circumstances.

Warnings

Already logging (W8)

You started a new log file when you were already logging.
MacDOS closes the old file and opens the new one.
Suggested actions:

- You can avoid this message by first closing the current log file with the command LOG without parameters.

Argument[s] not expected (W7)

You passed parameters to a command which accepts none. This warning is used when the execution of the command is not likely to have dangerous consequences. Generally, this mistake causes the abortion of the command and is reported as an error rather than a warning.
MacDOS ignores the parameter[s] and executes the command.
Suggested actions:

- Get rid of the parameters. Currently, only the commands PAUSE and SHIFT cause this warning message to be displayed.

DOS switch not implemented (W1)

You typed a switch used by DOS but not implemented in MacDOS. MacDOS ignores the switch[es] and executes the command.
Suggested actions:

- Check the list of available switches and stop using switches which are not implemented. You can also type `alarm on` and then repeat the command. MacDOS will then display the offending switch in the alert box used to report the warning.

Empty label (ie. colon not followed by a label name) (W6)

You typed a colon (ie. the character ':') on its own in a line of the batch program you are running.
MacDOS ignores the line.
Suggested actions:

- Remove the offending line from the batch file.

File not open (W4)

MacDOS attempted to close a disk file which was not currently open.
Suggested actions:

- This situation should never occur, as MacDOS should never close twice the same file or attempt to close a file which was never opened. Please try to reproduce the situation and, if you succeed, inform Rainbow Hill.

Illegal redirection (W5)

You redirected the input or the output of a command which should not be redirected.
MacDOS ignores the redirection and executes the command.
Suggested actions:

- Do not redirect the I/O of commands which do not support it. Refer to *Redirection and Piping* in the section "The Environment" of this User's Guide for the list of commands which accept redirection.

MacDOS warning ... (the dots are here used to indicate a number)

A warning which provides a code instead of a message in clear should never occur. The simplest explanation is that the resource containing all warning messages was not updated when the new code was introduced.
Suggested actions:

- Please report this occurrence to Rainbow Hill. Try to provide as much information as possible about the circumstances.

Not currently logging (W9)

You typed the command LOG without parameters while you were not logging.

Suggested actions:

- If you want to log, you have to type the command LOG followed by the name of the log file you want to open.

Request ignored (W3)

You tried to perform a mouse or menu operation which is not legal.

Suggested actions:

- Currently, this warning message is only displayed when you try to cut or copy particular portions of the console window. Type the command `alarm on` and repeat the offending operation to have a more explicit error message.

Sorting problem (W2)

When executing a `DIR` command, MacDOS attempted to perform a sort of a type not implemented.

The sorting is not performed.

Suggested actions:

- This situation should never occur, as MacDOS filters sorting options before attempting to execute them. Please, inform Rainbow Hill of the command which caused the problem.

Text line truncated (W10)

MacDOS has attempted to display a line which exceeds 255 characters in length.

Only the first 255 characters are displayed by MacDOS.

Suggested actions:

- You are probably listing a file with the command `TYPE`. If you can, edit the file and break the offending line into two.

Code Breaker

E1	DOS command not implemented
E2	Dimensioning: too many tokens
E3	File not opened by the user
E4	Volume offline
E5	Dimensioning: too many commands
E6	Dimensioning: too many arguments
E7	Syntax: double input redirection
E8	Syntax: double output redirection
E9	Line parsing error
E10	Internal: unknown com state
E11	Not a directory
E12	Syntax: invalid format
E13	Syntax: invalid arguments
E14	Invalid prompt switch
E15	Dimensioning: prompt too long
E16	Not enough space in the heap
E17	Error when following a path
E18	Maximum number of volumes exceeded
E19	Wrong volume specification
E20	Maximum number of directories exceeded
E21	Path with two consecutive backslashes
E22	The file is not executable
E23	Directory name missing
E24	MacOS error
E25	Internal: unknown token
E26	Max length of filename exceeded

E27	File or directory not found
E28	Bad switch
E29	Too many sorting requests
E30	Too many levels of batch CALLs
E31	Memory mode is 32-bit, but application is not 32-bit clean
E32	Attempted to open too many files
E33	Internal I/O error
E34	Illegal wildcarding
E35	(file not found)
E36	User Cancelled
E37	Too many characters in command line
E38	MacDOS requires System 6.0.4 or greater
E39	The pasting of newline characters into the command string is not allowed
E40	Not a file of type `'TEXT'`
E41	Bad command or file name
E42	(Do not report this)
E43	Command not implemented
E44	Label not found
E45	Too many labels
E46	Command only executable in batch
E47	Invalid date or date format
E48	Duplicate file name and version
E49	File busy, dir non empty, or path error
E50	No such volume
E51	Hardware volume lock
E52	Invalid time
E53	Directory specified
E54	(wrong file creator)
E55	(wrong file type)
E56	I/O error in disk operation
E57	Mismatch when verifying data written to disk
E58	Duplicate filename(s)
E59	Label used for more than one loop
E60	Volume literal ID not assigned
E61	The startup volume cannot be unmounted
E62	Nested FORs are not allowed
E63	Requested variable has not been defined
E64	No help available
E65	Labels must not begin with a percent sign
E66	Variables used in iterations must be unique
E67	Licence number and/or user identification corrupted. Please reload MacDOS from the original disk.
E68	Not a licence number
E69	Not enough memory
E70	Filter error
E71	Source and destination must be different
E72	String search failed
E73	Folders named Desktop Folder and Trash cannot be created or removed
E74	File open for writing
E75	File open for reading
E76	Maximum line length exceeded
E77	File already open for writing
E78	Standard FOR cannot execute an IF command
E79	The command cannot be piped
E80	Disk full
E81	The character requested for quoting is not free
E82	Sorry, the operation you have requested needs System 7.0 or greater

E83	The path contains too many characters
E84	Port does not exist at destination
E85	Invalid session reference number
E86	Session was closed
E87	Wrong destination in pipe message
E88	(PPC Operation not completed)
E89	Unknown pipe command
E90	Double initialisation of pipes
E91	(Data message correctly received)
E92	Filter sent a command that only MacDOS should issue
E93	Filter sent a data message out of a sequence
E94	Response addressed to a filter
E95	PPC Port already open. Restart the Mac if you need piping.
E96	Labels are only allowed in batch mode
E97	Remote process does not respond
E98	This Mac does not support PPC
E99	Unexpected data message from a filter
E100	Filter sent upstream a message that should only be sent downstream
E101	Filter sent downstream a message that should only be sent upstream

W1	DOS switch not implemented
W2	Sorting problem
W3	Request ignored
W4	File not open
W5	Illegal redirection
W6	Empty label (ie. colon not followed by a label name)
W7	Argument[s] not expected
W8	Already logging
W9	Not currently logging
W10	Text line truncated

Appendix B

Extension Programming

B Extension Programming

Introduction

With release 2.0.0, MacDOS™ supports a new and powerful piping mechanism. This allows programmers to extend the functionality of MacDOS through independent filter applications. The purpose of this section is to explain how the mechanism works and how programmers can make the best use of it.

To provide more than a simple cookbook, we have structured this section as follows: Functional Specification (what pipes are and what we expect from them); Design (how we implement pipes on the Mac); Programmer's Guide (the pipe toolbox and how to use it).

Functional Specification

In an operating system based on a command line interface (CLI), each command is usually associated with a process.

A pipe is a means of sending information from one process to another. Like water in a physical pipe, information in a software pipe only flows in one direction and what enters a pipe comes out of the other end in the same order (FIFO: First In First Out). The purpose of such a mechanism is to *feed* the second process with the output of the first one.

Predictably, another pipe can transfer information from the second process to a third one, still another pipe from the third to the fourth, etc. The end result is a chain of processes in which each process acts as a *filter*. Only the two processes at the beginning and at the end of the chain interact with the user (typically, via the keyboard and the monitor).

In principle, each process keeps checking whether the *incoming pipe* has something ready to be processed, does its processing on the bits and pieces it gets, and sends the result immediately to the *outgoing pipe*. In practice, it is the operating system that takes care of the piping. Each process still *believes* that it is getting its input from the keyboard and sending its output to the monitor.

As the processes are unaware that they communicate via pipes, all input/output is of ASCII characters rather than binary data.

The Mac OS is not built around a CLI and does not directly support I/O redirection and piping and the concept of standard I/O devices is missing.

Although MacDOS provides a CLI for the Mac, it operates at the application level. Therefore, its console window is not directly accessible to other applications. As a consequence, Mac applications (unlike UNIX® programs) must be *aware* of piping. Furthermore, as the piping mechanism is outside the OS, it is bound to be less efficient.

In view of all these considerations, we can specify piping for the Mac as follows:

- Piping shall be capable of transferring several characters in a single operation. This will reduce the overhead due to context switching between processes and to message handling within the processes themselves.
- The last filter of a chain shall send its output back to MacDOS, so that it can be displayed in the console window or redirected to a disk file.
- Any filter of a chain shall be able to report errors back to MacDOS, so that they can be displayed in the console window.

- The failure of any filter of a chain shall cause the quitting of all other filters and be detected by MacDOS so that it can be reported to the user.
- The user shall be able to terminate the operation of a chain of filters by typing a cntl-C/cmd-dot as with any other MacDOS command.
- MacDOS shall be able to pass to any filter its specific switches as typed by the user in the command line.

Additionally:

- The user shall be able to obtain on-line help by typing "`HELP filterName`" at the MacDOS prompt.
- MacDOS and the filters shall reside on the same Mac. That is, it shall not be possible to start MacDOS on system A and pipe a command to a filter which is on system B.

Design

This section describes how the generic functional requirements listed above translate into practice. That is, how MacDOS and filters exchange information.

General Strategy

Under the Mac OS, there are two basic ways for processes to exchange information: Apple Events (AEs) and Process to Process Communication (PPC).

Communication via AEs is *connectionless*, in the sense that each AE contains information about its destination process (a bit like sending letters through the mail). Also, when you send an AE, you have plenty of options on how you want the destination process to respond to it. This provides a lot of flexibility but introduces quite a bit of overhead as well.

Communication via the PPC toolbox is *connection oriented*, because you have to establish a communication session with the destination process before you can begin to send data to it, but then you can easily exchange messages without addressing them (a bit like making a telephone call). PPC requires some additional overhead at the beginning and limits flexibility later on, but it reduces the system overhead to do the actual data transfer.

In view of these considerations, MacDOS uses PPC to implement the pipes.

Before two processes can exchange information via PPC, they have to take the following steps:

- Each process opens a communication port (say, process 1 opens port A and process 2 opens port B).
- One of the processes (say, process 1), tells the OS that it is ready to accept PPC sessions.
- The other process (in this example, process 2) tells the OS that it would like to start a session with port A.

Normally, PPC operations are conducted asynchronously. That is, you start an operation by executing a function which returns immediately. To know when the OS has completed the operation you have requested, you have two possibilities: either you provide the OS with a *call-back* function (which the OS executes upon completion of the operation), or you check a flag.

MacDOS checks the flag, mainly because it is a simpler (ie. safer) mechanism.

As PPC sessions are bidirectional, filters can easily report back to MacDOS local errors and errors detected when communicating with the next filter.

Data Structures

Each PPC message contains two fields which can be used to store information:

- A long integer (4 bytes) called “userData”.
- A buffer called “dataBuffer”.

MacDOS uses the four bytes of userData as follows (in order of increasing memory address):
type specifies the content of the userData field.
seq is a sequence number set by MacDOS to identify each message.
src stands for **s**ou**rc**e and identifies the process which generates the message.
dst stands for **d**e**st**ination and identifies the process which should operate on the message.

dataBuffer always contains a P-string (ie. an unsigned character array with its first byte set to the number of characters that form the body of the message).

The possible message types and what they mean are listed in the table at the top of the next page.

Procedures

A filter needs to:

- Initialise the piping mechanism (open the PPC port, establish a session with the preceding filter, and initialise state variables).
- Process every incoming message.
- Forward to the next process every data message filtered.
- Report errors to MacDOS.
- Clean up (most importantly, close the PPC port, that otherwise remains open even after the filter has quit).

Type	Corresponding content of dataBuffer
pipeDataMess	To be filtered. This is what we use pipes for.
pipeParmMess	Filter configuration parameters (ie. switches).
pipeNextMess	MacDOS uses this message to direct a filter to establish a session with the PPC port of the next filter of the chain. It consists of the port name followed by the ID that MacDOS has assigned to the next filter.
pipeErrMess	The first byte is the Most Significat Byte of an error code of type OSType. The second byte of the dataBuffer is the Least Significant Byte of the same error code. When the error code is pipeFilteringErr, the following bytes of the message (if any) are characters used by the source filter to send to MacDOS an error message in clear.

MacDOS establishes the chain of filters as follows:

1. It opens its own PPC port.
2. It launches the first filter application.
3. It establishes a PPC session with the resulting process (for this to be possible, each filter must open a PPC port named like its own application file). When establishing the session, the userData field contains the ID of the filter (typecast from char to long).
4. It configures the filter by sending its paramaters as specified in the command line (pipeParmMess).
5. It launches the second filter application.
6. It directs the first filter to start a PPC session with the resulting new process (via a pipeNextMess to the first filter).
7. It configures the second filter via a pipeParmMess like it was done for the first filter.
8. It repeats steps 5 to 7 for all following filters in the chain until the last filter is configured.

9 It directs the last filter to start a PPC session with MacDOS.
10 It accepts the session opened by the last filter.

At this point, the full chain is completed and the actual processing of data can begin (via messages of type pipeDataMess). The message types mentioned so far only travel downstream (ie. from left to right in the chain of filters as they appear in the command line), while error messages only travel upstream (ie. back to MacDOS through all intermediate filters in the reverse order).

Each filter is connected to two sessions. The session with the preceding process (upstream) implements the incoming pipe, while the session with the following process (downstream) implements the outgoing pipe.

A filter only analyses and acts on messages which have as destination the ID of the filter. All other messages are forwarded to the other session unchanged. This is how MacDOS can build the chain one filter at a time and how filters can report errors back to MacDOS.

If you are familiar with concepts of Data Communication, the PPC represents a data-link protocol layer (OSI level 2), while the way in which MacDOS uses the userData field of the messages implements a network protocol layer (OSI level 3).

As it will become clear in the Programmer's Guide section, all configuration and forwarding operations can be "hidden" inside a single polling function, so that the programmer of a new filter does not need to be concerned with them. The basic structure of a filter is shown in Fig B1.

In most cases, MacDOS generates a data message every time a CR-terminated line of text is completed. When a filter receives a data message, it processes the message, formats the result of the processing into another pipeDataMess, and sends it to the next filter. When MacDOS receives a message from the last filter, it displays or stores it into a file as requested by the user in the command line.

The source-destination pair of data messages always identifies adjacent processes. That is, data messages are always processed after crossing a single pipe.

To avoid that MacDOS remains "hanging" (until it times out), filters must always forward a data message to the next filter. If, as a result of filtering, a filter has nothing to forward, it must send a message with a zero-length string in its dataBuffer. On the other hand, nothing prevents filters from sending several data messages to the next process as a reaction to a single incoming data message.

It is particularly important that filters send an empty data message to the outgoing pipe when they receive one from the incoming pipe. The reason is that MacDOS sends an empty message to flush the filter chain before prompting the user for a new command.

When a filter is waiting for incoming messages, it normally checks both the incoming and outgoig pipes. Nevertheless, after sending an error message to MacDOS, it only checks the incoming pipe and quits when it discovers that the session has been closed.

The Polling Startegy in Detail

The two PPC functions used to communicate through PPC sessions are PPCRead and PPCWrite. The filter shell included in the MacDOS release uses both functions asynchronously. DO NOT MODIFY ANY VARIABLE USED IN AN ASYNCHRONOUS PPC-CALL BEFORE THE OPERATION IS COMPLETED. If you do, you will cause unpredictable system errors and will have to restart your Mac.

A filter keeps an outstanding asynchronous PPCRead for each started session. This means that as soon as it receives a message from a pipe, it saves the message and immediately initiates a new Read operation.

When checking whether a new message has arrived (see the PipePollOnce function described in the next section), a filter always checks the outgoing pipe first. This gives priority to error messages (which travel upstream). The filter then keeps checking each pipe in a round robin fashion. After checking a pipe, the filter executes WaitNextEvent so that the Mac OS can activate other processes.

To send a message to a pipe (data messages to the outgoing pipe or error messages to the incoming pipe) a filter executes an asynchronous PPCWrite. It then only checks for the completion of the PPCWrite before sending the next message to the same pipe.

Typically, a filter spends most of its time polling the pipes and processing data messages. As explained above, the pipe polling generates a lot of calls to WaitNextEvent. If you type a cntl-C (or a cmd-dot), Wait NextEvent returns successfully and the filter sends to MacDOS the error code userCanceled. In fact, keyboard events are the only events that a filter handles.

Handling of the Sequence Number

MacDOS uses the **seq** field of the userData to decide whether it can send a new message and whether it can prompt the user for the next command. To achieve this result, MacDOS stamps all outgoing messages with ever increasing sequence numbers. As sequence numbers are stored in 8 bits, they wrap around 256 (ie. they restart from 0 after reaching 255).

The Mac OS ensures that messages are delivered in the same order in which they are sent, but MacDOS checks the sequence of incoming messages and aborts the command when it receives a message with a sequence number lower than that of the previous message. This is a safety precaution to protect MacDOS against some effects of badly designed filter applications (they could corrupt the userData of messages).

When sending a data message to the outgoing pipe, a filter uses the sequence number of the last message received from the incoming pipe. Therefore, if a filter responds to a single incoming message by sending several messages to the next filter, all those messages have the same sequence number.

Before sending a data message, MacDOS always updates the sequence number. Then, after sending the message, MacDOS expects to receive from the incoming pipe the filtered data message with the same sequence number. Only after receiving such a message, can MacDOS resume execution of the original command (this is why filters must always forward something).

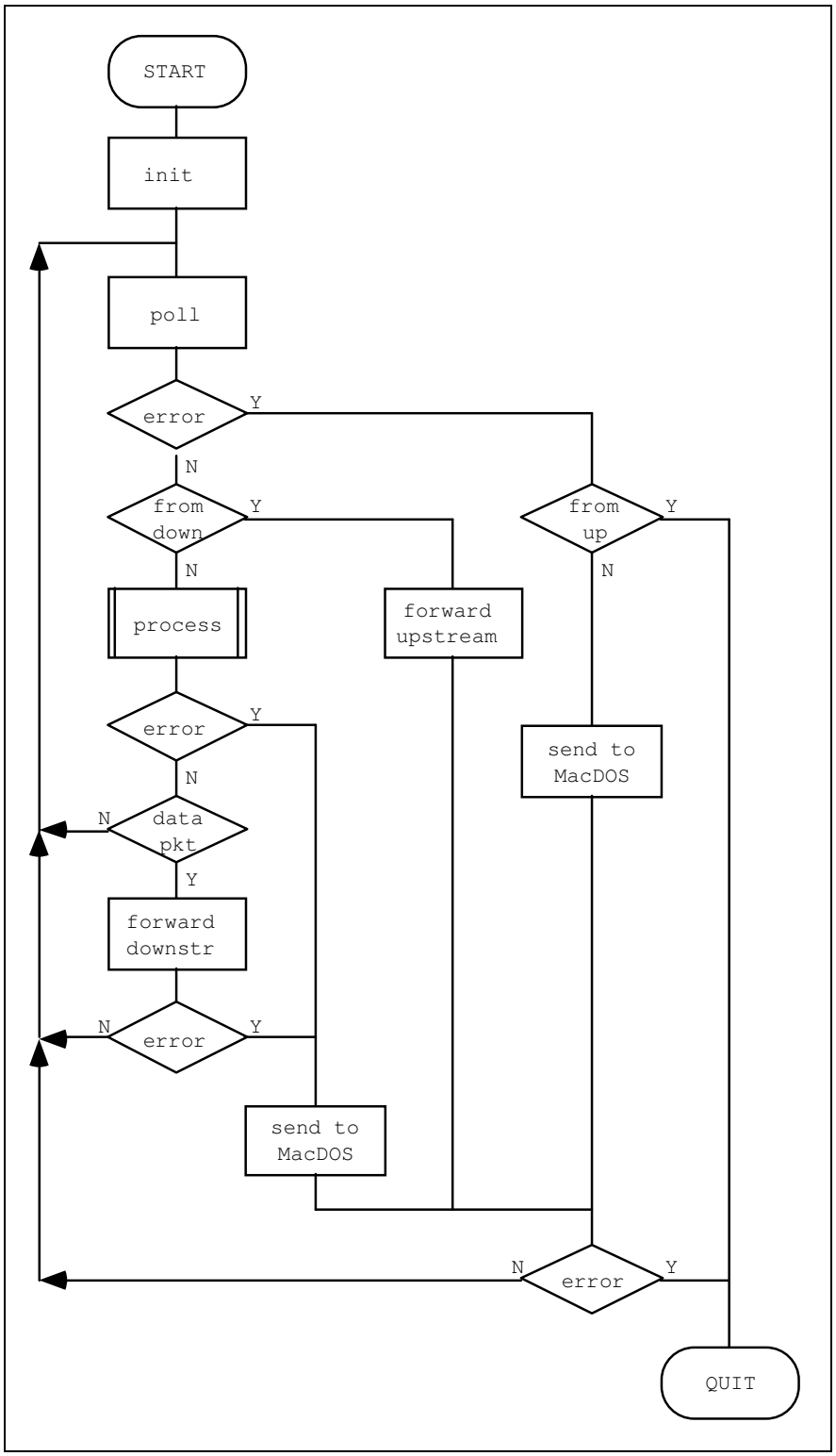

Fig B1: Filter Structure

Now, when a filter sends several messages with the same sequence number, MacDOS does not realise it until it begins waiting for the reply to its next message. In that case, MacDOS keeps processing all the incoming messages until it receives a message with the expected sequence

number. In this way, unless a filter "misbehaves", MacDOS avoids the piling up of incoming messages.

After completing execution of the original command, MacDOS sends a data message with an empty dataBuffer. This effectively "flushes" the chain of pipes and ensures that no filter-generated messages are outstanding. Only after receiving the corresponding empty message from the incoming pipe, does MacDOS close the PPC port and prompts the user for a new command.

Data Flow

This section illustrates in a graphic form how messages travel through the chain of filters. For this purpose, a chain of two filters is used. In all the following diagrams, messages travelling downstream are represented by arrows pointing to the right.

Fig B2 shows how MacDOS sets the chain up.

The dashed lines correspond to interactions with the PPC other than for sending messages. Each number IDentifies the destination filter, both for messages and for other actions.

Note that each filter performs a PipeInit (which includes accepting incoming PPC session requests) immediately after launch, while MacDOS only accepts sessions after directing the last filter to start one.

Once the initialisation is completed, both filters are in their default state, waiting for data messages.

Fig B3 shows how the normal data transfer looks in a simple case.

After sending a data message, MacDOS waits for the corresponding data message from its incoming pipe. Note that both filters are in their default state before receiving the data message and after completing its processing: they poll the pipes waiting for something to do.

Fig B4 provides an example of data transfer in which a filter sends two messages for each message it receives.

Note how MacDOS polls again after displaying 1b. It does so because MacDOS has already sent a message with sequence number 2, while 1b has still sequence number 1. MacDOS will only see 2b while waiting for 3, etc. After sending the last line with, say, sequence number N, MacDOS will send an empty message with sequence number N + 1, so as to detect possible Nb, Nc, etc.

Fig B5 shows how filters report error messages back to MacDOS.

The filters quit when they find out that their incoming pipe has disappeared (ie. that the incoming session is no longer there). Therefore, as soon as MacDOS closes its session with the first filter, that filter quits. This causes the disappearence of the next pipe, etc. This cascading effect terminates when the last filter quits.

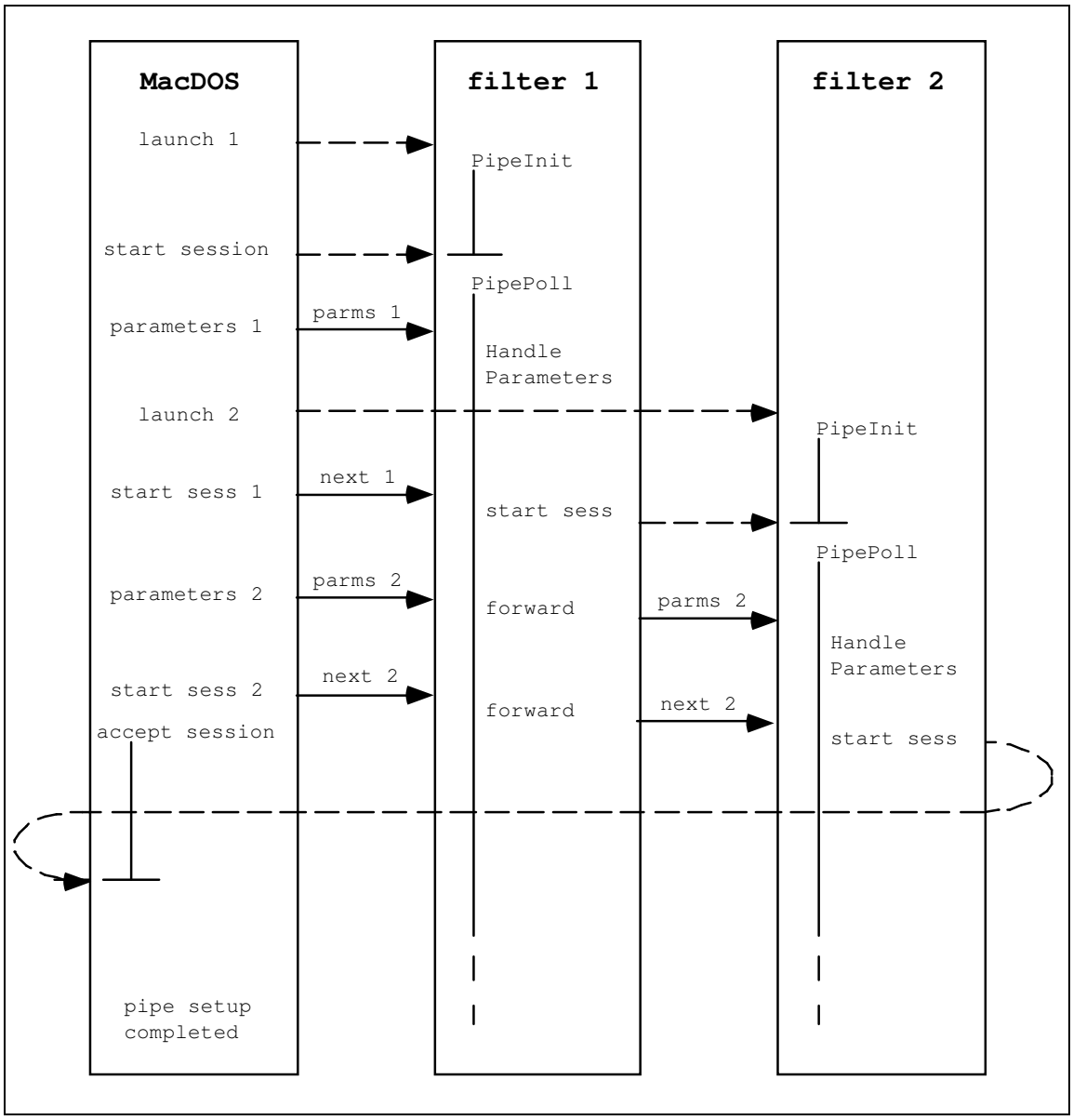

Fig B2: Pipe Set up

Programmer's Guide

This section describes the functions provided in the "filter shell" project. It also tells you how to structure and build a filter application.

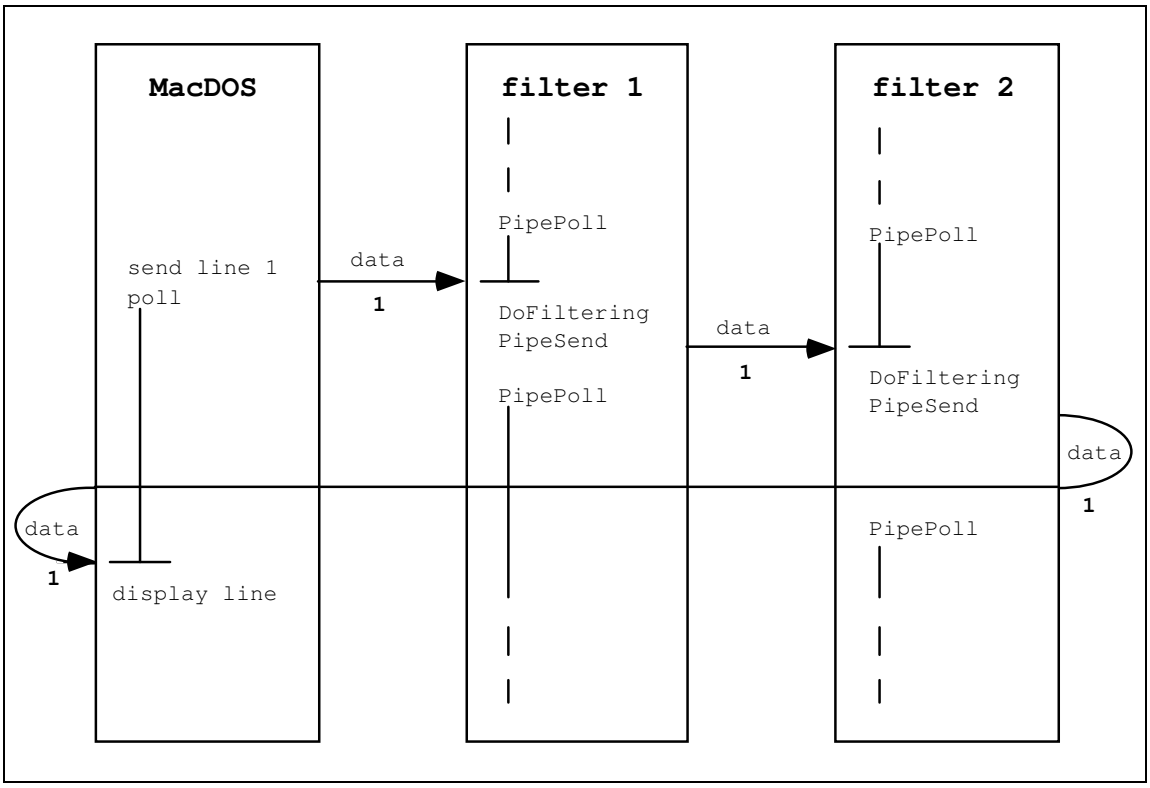

Fig B3: Pipe Transfer

Pipe Toolbox

The pipe.c/pipe.h sources provide the following functions (in alphabetical order):

```
void PipeInit(pipeParmsFun_t *parmsFun);
```

PipeInit prepares the filter to handle pipe messages. It opens the PPC port and only returns after establishing a session with the process upstream.

parmsFun is a pointer to a function which accepts configuration parameters as they appear in the command line and saves them in variables accessible during filtering. Set it to **nil** if your filter is not configurable.

```
void PipePoll(unsigned char *inMess);
```

PipePoll waits for messages from either PPC session. It automatically handles all messages unless they are data messages received from the incoming pipe and directed to the filter itself.

inMess is the pointer to the buffer where PipePoll can store an incoming message. inMess should be at least **pipeSize** byte long and will contain a P-string when PipePoll returns successfully.

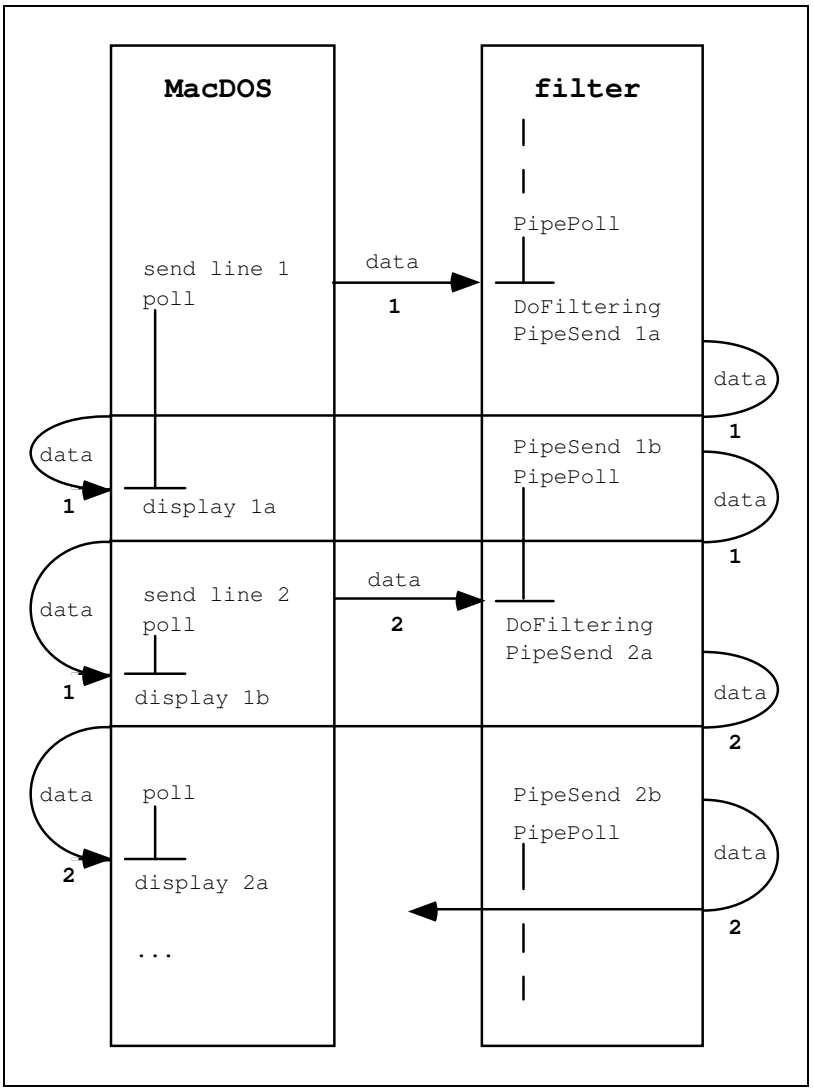

Fig B4: Multiple Messages

```
void PipeReportError(
                                OSErr               err,
                                unsigned char *errInClear
                                );
```

PipeReportError sends an error message to MacDOS. It can be used to report errors encountered during filtering.

err is the error code. To have MacDOS display a proprietary error string, set err to pipeFilteringErr. MacDOS will then display errInClear.

errInClear is a proprietary error string that you would like MacDOS to display in the console window. MacDOS only looks at this string if you set err to pipeFilteringErr. If you use PipeReportError to report a Mac system error rather than a proprietary error, you can set errInClear to **nil**.

```
void PipeSendData(unsigned char *outMess);
```

PipeSendData sends a message to the next filter downstream.

outMess is the pointer to the buffer containing the message to be sent. outMess must contain a P-string and cannot occupy more than **pipeSize** bytes.

Note that the Pipe functions only return if they are successful. Otherwise, they automatically attempt to report to MacDOS the first error they encounter. They then wait for MacDOS to "cut" the pipes.

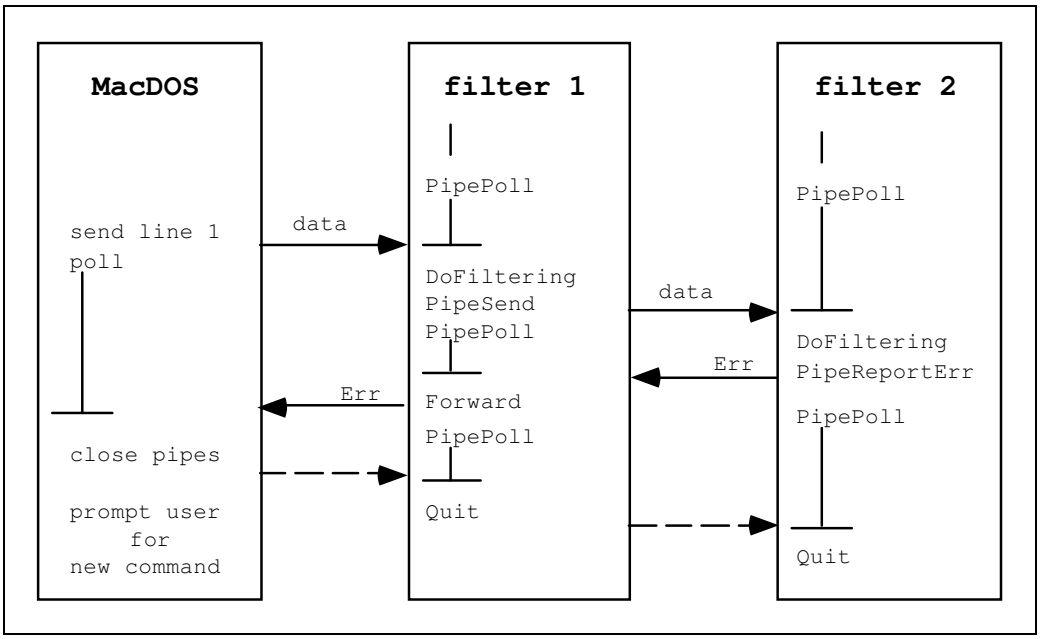

Fig B5: Error Messages

NEVER "DROP OFF THE BOTTOM" OF A FILTER. Let the Pipe functions take care of quitting the filter application. This will ensure that the filter closes its PPC port before quitting. If the PPC port remains open, you will not be able to re-use the same filter unless you restart your Mac (or change the filter application name, although this technique would lead to a proliferation of "ghost" ports. Only do it in emergency).

Filter Structure

The "filter shell" hides most of the complexities of a filter to the programmer. The resulting simplified structure of a typical filter is shown in Fig B6.

Debugging Filter Projects

In order to debug filters, you have to know how to do two things:

1 Launch your filter project instead of a compiled filter application.
2 Have MacDOS wait for you when you pause execution of the filter project with a breakpoint.

The first issue exists because MacDOS rightly recognises that your filter project is not a filter application. It expects a file of type `'APPL'` and creator `'mFLR'` but it finds a file of type `'PROJ'` and creator `'KAHL'` (only if you are using THINK C, but the discussion remains valid for other compilers). To trick MacDOS, do as follows:

- Place an alias of the file `fakeFilter.•` in the folder where you keep MacDOS.
- Rename the alias like your filter project.
- Run your filter project, set the breakpoint[s], and let it go.
- Click in the MacDOS console window to push the filter process to the background and bring MacDOS to the front.

- Type the command that you want to use to test your filter, but only type the name of your filter project, without its real path. MacDOS will see the renamed fakeFilter and will happily launch it (it will die at once). MacDOS will then succeed in opening a PPC session with the filter port, because your project was already running and had already opened the port correctly.

This trick works because the name of a filter application coincides with the name of the PPC port opened by the filter.

The second issue exists because MacDOS normally waits a couple of seconds for data messages to be looped back through the filter chain. If you pause execution of the filter with a breakpoint, MacDOS will timeout and abort the command. The solution is simple because MacDOS sets the timeout to the number of seconds specified in the global variable TIMEOUT. Therefore, if you type `"set timeout=3600"` at the prompt, MacDOS will wait for one hour before timing out.

Building a Filter Application

The simplest way to build a filter is to duplicate and rename the files `filter.c`, `filter.•`, and `filter.•.rsrc`. Within `filter.c`, you only need to modify the content of the function `doFiltering` (and `HandleParameters` if you want to use switches in the command line).

The file type of filters is `'APPL'` (no `'APPE'`, please), and their creator is `'mFLR'`. The fact that all filter applications have the same creator should not pose any problem, because MacDOS looks for them by name, and no document file type is defined for the same creator. Therefore, as you will not be able to start a filter by double clicking on a document, the Mac OS will never be placed in the position of having to choose a filter application on the basis of its creator string.

The project `filters.•` gives you some examples of how to define filter parameters and forward multiple messages to MacDOS.

The `HELP` command of MacDOS now supports on-line help of filter applications. What you need to do in order to set up an help message for your filter is to include a resource of type `'TEXT'` named “help” in its resource fork (the name “help” is not case sensitive). If you open `filter.•.rsrc` with resEdit, you will see that it already contains the default help message: *Sorry, no help available for this filter*.

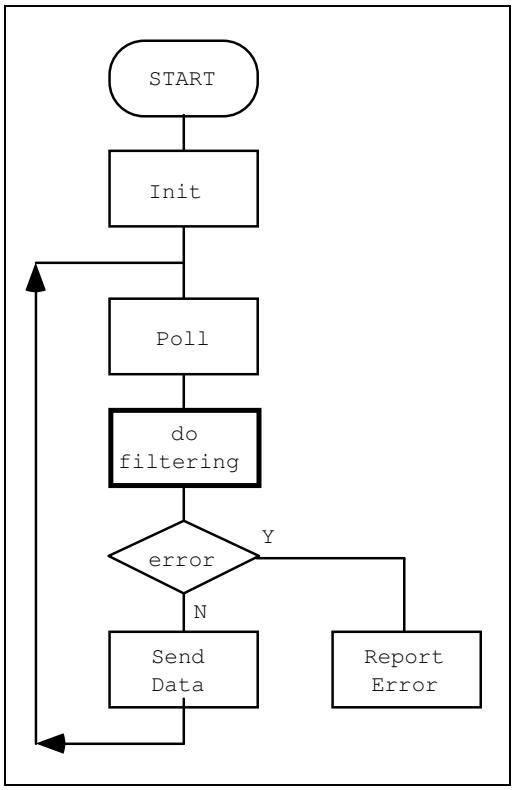

Fig B6: Simplest Filter

Abbreviations

AE	Apple Events
ASCII	American Standard Code of Information Interchange. This is the encoding of characters chosen by IBM, Apple, and most other computer manufacturers. The standard encoding assigns a value between 0 and 127 to 128 characters. Apple defines 128 additional characters to cover all 256 possible combinations of 8 bits.
CLI	Command Line Interface.
CR	The ASCII character Carriage Return. This is how the Mac OS separates lines of text.
DOS	Disk OS (most common Operating System for PCs).
FIFO	First In First Out
IBM	Industrial Business Machines (although they have recently changed the meaning of 'IBM')
ISO	International Organisation for Standardisation
OS	Operating System
OSI	Open System Interconnection. ISO's layered model for communication protocols.
PPC	Process to Process Communication toolbox
RH	Rainbow Hill Pty Ltd.

Appendix C

DOS Glossary

C DOS Glossary

This section uses Macintosh concepts to explain terms that MacDOS has inherited from DOS.

batch

Many Macintosh programs have their own scripting language. A batch program is nothing else than a script for MacDOS: you prepare a file containing a series of commands and then tell MacDOS to execute it.

As MacDOS supports a character oriented interface, batch and interactive commands are identical. This is not the case in most (if not all) other Macintosh programs, because they are normally controlled via dialogs and menus, while their scripting languages, by their own nature, consist of written directives.

Another way of understanding batch programs is to compare them with macros. When you prepare a macro, you switch the macro recording ON, perform some operations, and stop the recorder. By executing the macro, you can then repeat the same operations over and over again. The batch mechanism can work in exactly the same way: you start logging with the command `LOG`, type some commands, and stop the logging. By executing the log file as a batch program, you can then repeat the same commands as many times as you like.

cntl-C

By pressing the '`C`' key together with the control key, you force MacDOS to prompt you for a new command. This is useful when you want to interrupt a long `DIR`ectory listing or any other operation. cntl-C also interrupts batch programs.

If your keyboard does not have the control key, you can achieve the same result by pressing the command and period keys together (cmd-dot).

In some cases, especially during recursive operations, you might have to type cntl-C (or cmd-dot) more than once in order to interrupt MacDOS.

command

A command is just a way of telling MacDOS all it needs to know in order to perform a particular operation. The first word in a command (ie. the command name) tells MacDOS what to do, while the other words (ie. command parameters and switches) specify on what it should operate and how.

A command with switches is a bit like a menu: when you select a menu, you can choose between different options by selecting a particular menu item. When you type a command in MacDOS, you can choose between different options by typing particular switches.

delete

The word "delete" always refers to files, never to folders. When you use the command `DEL` to delete files, MacDOS removes the corresponding entry from the disk directory. The result is that the file immediately "disappears" from the system and the space it occupied is made available for other uses. This is VERY different from the standard behaviour of the Finder, which only removes files from the system when you empty the Trash.

directory

"directory" essentially is a synonym of "folder". In fact, the term "folder" was only introduced together with the desktop metaphor. By then, UNIX users were already familiar with commands like `CD` and `MKDIR`.

Sometimes, the term "directory" is used to indicate a "directory list" produced with the command `DIR`. This might not be entirely correct, but it does not normally create any confusion.

drive

Drives are physical devices which let you access storage media. To distinguish between different drives, operating systems number them. For example, the Mac OS assigns the number 1 to the internal floppy drive and 2 to the external one. DOS uses the letters A and B for the same purpose.

The "drive numbers" of the Macintosh and the "drive IDs" of DOS are basically the same thing.

extension

The Mac OS associates a series of parameters and flags to each file in the system. In that way, it can decide whether, how, and where each file icon should appear on the desktop. Just by looking at an icon, you can normally distinguish between applications and documents. Furthermore, you can also identify which application created each document.

DOS does not have the capabilities of the Mac OS. In order to distinguish between different file types, DOS splits each filename into two parts separated by a dot, and uses the second part to identify the type. That second part, which cannot exceed three characters in length, is called the "extension" (nothing to do with the MacDOS extensions, which are filter applications). Generic text files should have the extension TXT, batch programs BAT, applications EXE (for EXEcutable), documents DOC, etc.

This strategy, at least as it has been implemented, can cause quite a bit of confusion. This is due to the fact that DOS does not (and cannot) enforce the naming conventions. Therefore, you are totally free to use the extensions as you like. For example, you can call a text file `readme.doc` or an application `process.txt`. You can also create non-standard names like `use.me` or `this_is.it`. As if this were not enough, you cannot even distinguish between files created with different applications. For example, a file called `book.doc` could have been produced with any word processor.

file

In "Inside Macintosh Vol 2", Apple states: "A **file** is a named, ordered sequence of bytes". This is also applicable to DOS files. Both with DOS and the Mac OS you normally operate on files as single entities, although Macintosh files consist of two separate forks. In some cases, MacDOS lets you operate separately on Data and Resource forks.

global variables

Global variables have two main functions: to store temporary information during the execution of batch programs and to memorise your "preferences" if you set them within `autoexec.bat`.

parameter

In most cases, parameters identify the items on which commands and batch programs operate. Many commands expect file or folder specifications as parameters. Therefore, the passing of parameters to a MacDOS command often corresponds to filling in text fields in a Macintosh dialog box (eg. after selecting the "Open..." or "Save as..." menu items).

path

A path is nothing else than the list of folders you have to enter in order to reach a particular item (file or folder). The MacDOS convention is to use a backslash to separate adjacent folder names.

A path which begins with a backslash is said to be "absolute" and indicates that you have to start entering folders from the outermost one. A path which begins with a folder name (ie. not with a backslash) is said to be "relative", and indicates that your starting point is the current folder. The fictitious folder name ".." indicates that you have to enter the folder which contains the one you are currently in.

When you add a folder name to a path, it is as if you memorised the double-clicking on the corresponding folder icon.

pipe

A pipe is a way of using the output of a command as input to another command.

In Macintosh terms, the closest thing to a pipe is to "Cut" and "Paste" or "Drag" and "Drop" between different applications.

prompt

MacDOS displays the command prompt to let you know that it is ready to accept a new command. This is equivalent to the Finder changing the cursor from a watch back to an arrow.

Other prompts (eg. when asking confirmation before deleting a file) are equivalent to displaying dialog boxes like the "...Are you sure you want to permanently remove it?" alert that you get when you empty the Trash.

redirection

Many commands display information on the monitor screen. Some, like `DIR` and `TYPE`, do nothing else. Others, like `COPY` and `RENAME`, do it to inform you of their progress. In all cases, you can save that information for later examination and processing by redirecting the command output to a disk file.

Similarly, input redirection lets you prepare input data off-line and then use it when you need it, thereby freeing you from the task of typing it interactively.

As both output to the monitor and input from the keyboard consist of ASCII characters, files used for redirection are always of type `'TEXT'`.

In a sense, I/O redirection is similar to the "export" and "import" functions of many Macintosh applications.

root

The "root" is the filing cabinet where you keep all your folders. As there is exactly one root per volume (ie. hard disk partition, floppy, etc.), for all practical purposes you can identify the root with the volume itself.

When you format a floppy, the Operating System automatically creates the "root". It is then up to you to grow "trunk" folders and "branch" off into further subfolders. The "leaves" of such a directory tree are files, which you are not allowed to further subdivide (although the Macintosh still gives you two "forks"). In the Macintosh environment, the root is represented by the window that you get when you double click on a volume icon.

switch

A DOS switch is a part of a command which lets you enable or disable optional features of the command. In its simplest form, a switch consists of a slash followed by a letter of the alphabet, but switches which control powerful options are more complex than that.

The Macintosh equivalent of a switch is an option that you can select via a dialog box. For example, when you select the "Print..." menu item, you are presented with a dialog box which lets you select page ranges, number of copies, etc. In DOS, you would select those options by adding switches to the `PRINT` command.

wildcarding

Wildcarding is a way of selecting group of files on the basis of parts of their names. In most cases, you can operate on wildcarded filenames as you would operate on single files, leaving to DOS the task of repeating the same action on each individual file. The Macintosh Finder lets you "Find..." files with names containing a particular string, but then you still have to access the files with the mouse.

Appendix D

Command Index

D Command Index