# Pro-DOS

By Chris Pile

# PRO-DOS

Pro-Dos emulates the CP/M 2.2 environment on the Sam Coupe computer. A minimum configuration is a 256K machine with one disc drive.

Disc Format is 706K, 9 sectors/track, 256 directory entries.

Automatic drive allocation for different SAM configurations

61K Transient Program Area, 124K Ramdrive on 256K SAMs and 380K on 512K SAMs.

Resident Commands

DIR, ERA, TYPE, REN, USER, SAM, SAVE, CLS

Transient commands

PALETTE, FORMAT, BATCH, CRC, COPY, DUMP, SAMREAD, HIDE, UNHIDE, PROTECT, UPROTECT.

VT52 Terminal Emulation (80 column two colours and inverse video)

68 page manual is supplied with information for both the beginner and expert.

**Contents**

**PRO-DOS**
**(C) Digital Reality 1991.**






Please Note September 2003. - Revised Jan 2005

www.samcoupe-pro-dos.co.uk

This Pro-Dos Manual has been transferred via OCR Pocket Reader. It was transferred by Stephen Parry-Thomas for the following reasons.

To preserve the Pro-Dos manual from getting lost forever, and to restore software and manuals for future use by Sam Coupé user's enthusiasts, programmers.

While every effort has been made to contact the copyright owners for permission, at present no contact has been made so the restoration has taken place in the form of this document.

Contents

This page has been left intentionally blank

## Preface

I would like to take this opportunity to thank the following:

Brian Gaff. For lending ma a Sam Coupé in the first place! Also for bug-testing the final version. Oh, and for nagging me into improving the key-scan routines!

Wayne Weedon. For editing and producing the user manual, and bug-testing, not only the final version, but also during the development stage.

Pro-Dos is intended to be easy to use but flexible enough to build on, with Pro-Dos a whole world of serious and not-so-serious software becomes available to users of the Sam Coupe.

I hope you enjoy using Pro-Dos and it proves to be useful addition to your software collection.

Chris Pile.
October 1991.

## Introduction

**What is Pro-Dos?**

Pro-Dos could be thought of as a Disk Operating System (DOS), however, it is more than this, it is a full Operating System that provides compatibility with CP/M 2.2. See COMPATIBILITY note on page 06.

In simple terms this means that a whole world of software that was designed to run under CP/M 2.2. will now run on the Sam Coupé.

Pro-Dos uses the same Disk format as the Amstrad PCW 8256 and, as a result, can read disks from this machine direct, this also means that there is a vast range of software already available from sources such as Public Domain libraries.

When buying software from a Public Domain library you must check that it was designed to run under CP/M 2.2 or one of the lower versions, 1.4 for example. Some software will only run under the later versions of CP/M, commonly known as CP/M 3 or CP/M Plus. However, the majority of available software is 2.2 compatible and you should have no trouble running it under Pro-Dos.

Some software may need "installing" for the particular machine you intent to run it on, this normally involves terminal (or Screen) configuration and sometimes disk data information, most software includes full configuration instructions and it is usually a simple task.

Please refer to Appendix A for information on terminal escape codes and Appendix B for information on the Pro-Dos disk format.

Some Public Domain libraries require you to supply the disk format structure information, once again, refer to Appendix B.

Programmers wishing to write programs to run under Pro-Dos are referred to Appendices A,B,C, and D, these provide the terminal escape codes, system calls and memory usage information. As an alternative (and more detailed) reference to the system calls you are urged to read one of the many available books on CP/M 2.2.

Pro-Dos runs totally independent of the SAM ROM and DOS and, as a result, makes no ROM or SAMDOS calls. Therefore, Pro-Dos will run on any SAM with any version of SAM or Master Dos. Indeed, to boot Pro-Dos from its boot disk you do not need to have pre-loaded SAM or Master Dos at all.

Pro-Dos automatically adjusts itself to your particular machine, be it a 256k single drive, or a 512k twin drive system, system this ensures that it makes the best use of the RAM and drives available.

You are referred to Appendix C for full memory usage information and also a description of what Pro-Dos actually does after it has booted.

**COMPATTBTLTTY NOTE.**

Every effort has been made to ensure that Pro-Dos is software compatible with CP/M! 2.2. During the development of Pro-Dos over one hundred programs, ranging from simple text editors to complex Language compilers like "BORLANDS Turbo PASCAL (tm) ", have been tested and all worked successfully.

However, no guarantees can be made and there could be programs that, for some reason, do not function as they should.

If you come across such a program you should first establish that it was designed to run under CP/M 2.2, reading the instructions that come with it will probably tell you this. Whilst you are reading the instructions you should also check that you are doing everything correctly.

If the program was designed for 2.2 and you are doing everything correctly then check if the program needs to be installed for the particular machine you wish to run it on. If so then you will have to install it and try again. Most programs will include their own install program and instructions and normally it only involves; changing the screen (or terminal) escape codes.

Once installed does it function correctly? If not then it could be that the program is "misbehaving" in some way, possibly it is trying to modify part of the operating system itself, or it might have been designed to run on a specific machine and as a result makes system or hardware specific calls that do not exist in Pro-Dos, or in any other machine apart from the one it was designed for. An example here would be disk go formatting programs, these will almost certainly be specific to a particular machine. These types of program are best avoided.

When collecting programs try to ensure that you get the generic version, this way you can be almost certain that the program has not been ''patched'' for some other machine.

Pro-Dos differs from the standard CP/M 2.2 memory layout in the fact that the system itself remains hidden in another RAM page. This may cause compatibility problems 'with programs that try to modify the actual system in some way, as the system itself is not visible to a program.

Programs that do these "dirty tricks" are rare, in fact I nave yet to come across any. If you come across this type of program then its best to avoid it, because it will probably cause all sorts of weird problems when it tries to modify the system!

Again, check the programs instructions. If it mentions modifying ANY part of the system code then it’s best to avoid it.

All that said, however, you should have very few problems running software under Pro-Dos. Remember, there are thousands of programs available to run under CP/M 2.2!

Programmers wishing to write programs to run under Pro-Dos should also read the notes on interrupts in Appendix C, page 53.

# Introduction

# Chapter 1

## Getting Started.

Pro-Dos comes supplied on two 3.5'' disk, marked BOOT and SYSTEM FILES.

Due to the way Pro-Dos is booted you cannot make backups of the BOOT disk, the write protect tab has been removed to prevent accidental corruption. Several software error recovery methods are built into the boot disk, as a result it is very robust and you should not have any trouble with it, even if Part of it does become corrupt. However, should the disk become corrupt enough to stop Pro-Dos loading , (indicated by a psychedelic colour screen either during the load or after) then return the boot disk to the supplier where you purchased Pro-Dos, and a replacement will be arranged.

The SYSTEM FILES disk contains the Pro-Dos external commands, this disk is in the Pro-Dos format and you cannot use SAMDOS to make backups of it. However, you are urged to backup this disk to another Pro-Dos format disk using the external commands FORMAT and COPY, these are explained in chapter 4.

To load Pro-Dos insert the disk marked BOOT into Drive A: and press F9, remember, you do not have to pre-load SAM or MASTER DOS before booting Pro-Dos. After a few seconds you should see the Pro-Dos start-up screen, it will look something like this:

**Pro-Dos v1.x. Program By Chris Pile © Digital Reality 1991.**

**Available Drives: A B**
**RAM Drive - Drive B: Size: 124k**

**Insert A Pro-Dos Disk Into Drive A:, Then Press Space.**

The above is what you would see if you had a 256k SAM with a single floppy disk drive.

On a 512k SAM the RAM disk size increases to 380k.

On a SAM with twin floppy: drives the available drives would be A B C and the RAM drive would move from B to drive C. Below is what you would see if you had a 512k SAM with twin floppy drives:

**Pro-Dos v1.x. Program By Chris Pile © Digital Reality 1991.**

**Available Drives: A B C**
**RAM Drive - Drive C: Size: 380k**

**Insert A Pro-Dos Disk Into Drive A:, Then Press Space.**

You should now remove the BOOT disk and insert a valid Pro-Dos formatted disk into drive A:, the SYSTEM FILES disk will do. After Pressing SPACE you will see the "A>_ " prompt.

The next chapter deals with the prompt and the command line.

# Chapter 1

# Chapter 2

## The Command Line.

**THE SYSTEM PROMPT.**

If mentioned at the end of chapter 1, when you press SPACE from the start-up screen you will see the system prompt:

**A>_**

The prompt is divided into four parts, the current drive, the user area (not shown for user area 0, the default), the > symbol and the cursor.

So, drive A, user area 5 would look Like:

**A5>_**

Drive B, user area 0 would be:

**B>_**

The cursor is where the next typed character will appear.

**CHANGING DRIVES.**

From the start-up screen the prompt shows drive A, this is the default drive for all subsequent disk operations. You can change to any of your available drives by typing the drive letter followed by a colon.

E.G. From drive A you can move to drive B by typing B: followed by RETURN.

NOTE: Case is not important in the command line, b: would have been valid.

When you change drives you remain in the same user area as the previous drive. E.G. If you are in user area 5 on drive A and move to drive B your prompt would then look like this:

**B5>_**

More on user areas in the next chapter.

If you try to move to a drive that does not exist, you will be given the INVALID DRIVE error message. A full list of error messages and their meanings are contained in chapter 5.

If your SAM only has one floppy drive then drive B is a fast RAM disk on a twin floppy drive SAM drives A and B are the two floppy's and drive C becomes the RAM disk . The RAM disk is useful for small jobs because it is quick and quite large, (124k on a 256k SAM, 380k on a 512k SAM) however, you MUST remember to copy all files you wish to keep, from the RAM disk onto a floppy disk

before you switch your SAM off, else all the RAM disk files will be lost. Usage of the RAM disk is identical to the floppy drives and all explanations regarding the floppy drives are equally applicable to the RAM disk.

**ENTERTNG COMMANDS.**

ALL disk functions and the running of executable .COM files are performed from the command line. You enter commands by simply typing them, and pressing RETURN.

Several editing features are available whilst in the command line, they are listed below:

**CTRL-X CTRL-E CTRL-R CTRL-P CTRL-S CURSOR LEFT**

**CURSOR RIGHT RETURN DELETE**

CTRL refers to the SAM CNTRL key, CTRL-X means press the CNTRL and X keys simultaneously.

Each editing function is now explained:

**CTRL-X**

Eases everything to the left of the current cursor position, the cursor and everything to the right of it will shift left until it reaches the > prompt.

**CTRL-E**

Moves the cursor one character after the last character on the current command Line.

**CTRL-R**

Re-call the last command line entered by the user, erasing the current one.

**CTRL-P**

Toggles printer output, when this option is ON a capital P is displayed in the top right of the screen, and all screen output will the also be sent to the centronics port on the SAM COMMS interface.

**CTRL-S**

Toggles screen pauses, when this option is ON, a capital S is displayed in the top right of the screen, when the screen becomes full the system will pause, to indicate this the S will start flashing. When the system has paused, just press any key to resume, Pressing CTRL-S in the paused state will turn off this feature.

**CURSOR LEFT**

Moves the cursor left. This allows you to move back to a given position and insert or delete. If you move back and start to type, the new text will be automatically inserted and all the text to the right of the cursor will be shifted right. If you start to delete, any text to the right of the cursor is shifted left.

**CURSOR RIGHT**

Providing the cursor is not already at the end of the current command line will move the cursor right. Its purpose is the same as CURSOR LEFT, and by using these keys, you can freely edit and change the current command Line.

**RETURN**

Enters the current command line, which is then processed. NOTE: CURSOR DOWN has the same effect as RETURN.

**DELETE**

Deletes the character to the left of the cursor, if the cursor is not at the end of the current command line, all the text to the right will be shifted left.

All other CTRL keys typed at the command line, and all CTRL characters printed on screen are shown in inverse Lower case.

To load and run external programs or to execute any at the internal Pro-Dos commands you simply type it and press RETURN.

Remember, case is not important on the command line. Should you wish for upper case characters just hold dawn SHIFT and type. You can turn upper case on and off by pressing CAPS. When in CAPS mode the cursor changes from an underscore to a solid block.

NOTE: The command line has a maximum length of 70 characters, when this limit is reached you will not be able to type (or insert) any more characters. DELETE, RETURN, CTRL-X Etc. are still available.

**FILENAMES.**

Pro-Dos uses the same filename structure as CP/M. This may already be familiar to you, if not it is detailed below:

Filenames can be thought of as two parts, the actual filename and the file extension. The filename part can be up to eight characters long and the extension up to three, a full stop (or period) is used to separate the two parts, and not all filenames have extensions.

The following are all valid Pro-Dos filenames:

READ.ME  RUN.COM  SOMEDATA.DAT  HELLO  TEST. ASM

Filenames that have the .COM extension (like RUNME .COM, above) are executable programs, these could be anything from a game of Chess to a word processor .COM, as an extension, can be thought of as an abbreviation for command. As a result, COM files are treated like commands, and will run automatically once loaded from disk.

NOTE: you do not have to include the .COM part of the filename if you wish to run a particular program, just enter it's name (like RUNME) from the command line, it will then Load and run.

You can also prefix a filename with a drive letter followed by a colon:

CHESS.COM

Would indicate the file CHESS.COM on the current default drive and user area, as indicated by the system prompt.

B: CHESS.COM

Would indicate the file CHESS.COM on drive B, user area as shown in the system prompt.

**WILDCARD.**

Most of the Pro-Dos internal commands, and indeed, most external .COM programs allow the use of Wildcards in filenames. There are two Wildcard characters available, they are * and ? . The ? character will match any single character at that particular position in the filename.

?ELLO .COM will match:

HELLO.COM  MELLO.COM  YELLO .COM  FELLO .COM

But will not match:

HELLO.TXT  MELLON.COM  YELLOW.CON  FELLO.DOC

The * character will match all remaining characters in either the filename or extension fields.

*. COM will match:

HELLO.COM  MELLO COM  YELLO.COM  FELLO.COM

But will not match:

HELLO.DOC  MELLO.ASM  YELLO.TXT  FELLO .BAT

?E* .* will match:

HELLO.COM MELLO.COM YELLO .COM FELLO .COM

It would also match:

HELLO.DOC MELLOW.ASM YELLOW.TXT FELLOW .BAT

Finally, *.* will match ANY filename.

If any of the Pro-Dos internal or external commands will not accept Wildcards then the explanation of that command will state this.

The following characters cannot be used in filenames: < >,:=;[ ]

The next chapter deals with the internal commands and their syntax.

Chapter 2

# Chapter 3

## Built-in Commands.

**INTERNAL COMMANDS.**

Pro-Dos provides 12 internal (or built-in) commands.

The first four are disk directory commands, they allow you to view the files that exist on your disks. The four commands are basically the same and similar screen output. Each can be followed with a filename argument and they all share the same Syntax.

They are: DIR, DIRS, DIRN and DIRNS

**DIR**

This is the standard disk directory command and the one that you will probably use the most.

Used on it's own it will give you a directory of the disk in the current drive and user area. It might look like:

```
 A>dir        (Remember to press RETURN after issuing a command.)

A:WORPRO  .COM | EXAMPLE .DOC | KEYS    .CTL | READ    .ME
A:HELLO   .    | TEST    .COM | ASSEMBLE.COM
7 File(s).  Free Disk Space 582k

A>
```

After the list of files on the disk You are shown how many files there are and also how much free space is left on the disk, in kilobytes. The system then returns to the command line ready for the next command.

If you would like a printout of the directory you can issue a CTRL-P before the DIR.

You can choose to specify a Particular filename or filenames (using wildcards). Some examples:

```
A>dir *.com

A:WORDPRO .COM  |  TEST       .COM  |  ASSEMBLE .COM
3 File(s).  Free Disk Space 582K

A>dir  test.com

A:TEST     .COM
1 File(s).   Free Disk Space 582k
```

Remember, you can prefix filenames with a drive letter followed by a colon, so you could look at drive B without actually moving there:

```
A>dir b:

B:EXAMPLE.DOC | WORDPRO.COM
2 File(s). Free Disk Space 54k

A>
```

**DIRS**

This is basically the same as the DIR command, the screen (or printer) output is identical. This command differs from the standard DIR in the fact that it only shows files that have their SYSTEM flag set. These files are otherwise hidden. You can set in reset this flag by using the HIDE.COM and UNHIDE.COM programs respectively, these are external commands and are explained in the next chapter.

**DIRN**

This is the same as the DIR command, with the exception of screen or printer) output. Whereas DIR displays the files in five columns, DIRN displays just a single column. This is sometimes better for printouts and the above example directory would lock like this:

```
A>dirn

A:WORDPRO  .COM
A:EXAMPLE  .DOC
A:KEYS     .CTL
A:READ     .ME
A:HELLO    .
A:TEST     .COM
A:ASSEMBLE.COM
7 File(s). Free Disk Space.

A>
```

**DIRNS**

This is the same as the DIRS command with the screen (or printer) Output from the DIRN command.

The next three commands are the simplest, they take no additional arguments and do not reference the disk drives.

They are: SAM, CLS and FEEDS

**SAM**

Allows you to reset the SAM, it's result is the same as pressing the RESET button .

Once entered you will be given the option of aborting the reset.

**CLS**

Clears the screen if things are looking a bit messy.

**FEEDS**

Toggles the LINE FEEDS that are sent to your printer.

Some Printers only require carriage returns and if line feeds are also sent will cause double Line spacing.

Once entered, you will be told if line feeds will be stripped or sent, the system defaults to sending line feeds.

The next three commands are used to RENAME, SAVE and ERASE files on your disks. ALL three must be followed with the correct number of arguments, with the exception of the erase command.

They are: REN, SAVE and ERA.

**REN**

Allows the renaming of files on your disks and takes two filename arguments separated by one or more spaces. The syntax is:

**REN oldname.ext newname.ext** (No Wildcards Allowed)

The above example would rename the file "oldname.ext" to "newname.ext" in the current user area and drive. (As shown in the system prompt).

You can precede the first filename (the old name) with a drive letter.

**REN b:oldname.ext newname.ext**

Which would rename the file "oldname.ext" to "newname.ext" in the current user area on drive B.

The second filename (the new name) is automatically given either the current drive or the drive specified in the first filename, ignored.

If you try renaming a file to a name that already exists on that particular disk you will be given the ALREADY EXISTS error message.

**SAVE**

Allows you to save to disk, the contents of the current system memory, starting at address 256 (100 hex). It takes two arguments separated by one or more spaces. They are the number of 256 byte blocks to save and the filename. The syntax is:

**SAVE 123 test.dat** (No wildcards Allowed)

The above example would save 123 x 256 byte blocks (31488 bytes) from address 256, to the current user area and drive with the filename TEST.DAT

Yon must supply the number of blocks to save as a decimal number, this can be from 1 to 245 and the filename can be preceded by a drive letter.

If you try saving a file with a name that already exists on that particular disk you will be given the ALREADY EXISTS error message.

**ERA**

Allows the erasing of a file or files from your disks. It can take one (optional) argument in the form of a filename. The Syntax is:

**ERA filename.ext**

The above example would erase the file FILENAMEE.EXT from the current user area and drive. You can precede the filename with a drive letter, and Wildcards in the filename are allowed.

**ERA b: *. com**

Would erase ALL the .COM files in the current user area an drive B.

The filename is optional, if not supplied it is treated as an ERA *.*

**ERA**

Would erase ALL the files from the current user area and drive.

**ERA b:**

Would erase ALL the files from the current user area on drive B:.

If you enter ERA alone you will see the following system prompt:

**Erase *.*, Are You Sure Y/N**

Pressing ANY key except Y will abort the command.

Note: You cannot erase a file or files, if the disk is write Protected, if you try you will be given an error message.

You can write protect a disk by opening the write protect tab.

Sometimes a program will instruct the system to prevent writes to the disk, if a write then occurs the system will treat the disk as if the write protect tab is open, giving the same error message.

You can prevent individual programs or files from being erased by using the PROTECT.COM program as detailed in the next chapter. If you try erasing a file that is protected the ERA command will ignore it. You should remember that this will not protect them if you format the disk, the only way to Protect against a disk format is to open the write protect tab.

The final two internal commands are used to change user area and to display text files.

They are: USER and TYPE

**USER**

Allows you to change the current user area. Its syntax is:

**USER nn** (Where nn is the user number from 0 to 15)

User areas are used to segregate files that reside on a single disk. For instance, you could have all your text editors in user area 0 and your assemblers in user area 1.

All disk operations are performed on the current user area, the one shown in the system prompt. Therefore, if you are in user area 1 you cannot erase or rename a file in user area 0. Likewise, you would not be able to run a .COM file from user area 15.

**TYPE**

Allows you to view files on screen and, optionally .end them to your printer, It is commonly used on text files and takes an optional filename argument I which can include Wildcards. It’s

syntax is :

**TYPE filename.ext**

The above example would display the file FTLENAME .EXT on screen in ASCIT form. You can precede the filename with a drive letter. If you do not supply a filename then *.* is assumed.

Using wildcards you can choose which files you wish to view.

**TYPE b: *.txt**

Will give you the option of viewing any or all of the files with the extension .TXT in the current 'user area on drive B. When Pro-Dos finds a filename match you will see the following:

**V to View: somename.TXT, S to Skip, Q to Quit.**

Pressing V will display the file, S will skip this file and try a find another filename match and g will quit back to the system Prompt.

During the displaying of a file you can press Q to quit, if you press Q and your original filename contained wildcards then Pro-Dos will try to find another filename match, if it did not contain wildcards, you are returned to the system prompt. You can also use CTRL-S and CTRL-P either before entering the TYPE command or during the actual display, to toggle screen pauses and printer echo respectively.

The next chapter deals with the Pro-Dos external commands.

# Chapter 4

## External Commands.

**EXTERNAL COMMANDS**

Pro-Dos provides 11 external commands. These are on the SYSTEM FILES disk as .COM programs.

These .COM programs are executed, (as are all .COM programs) by simply typing the filename (without the .COM), any arguments the program may require, must be separated from the filename with one or more spaces, then press RETURN.

You must however, ensure that the file resides on the disk in the current drive, Or the disk you specify in the filename, otherwise you will be given the "Can't Find:" error message

Any of the Pro-Dos external command programs that require additional arguments will display a simple "How to use" line if you just enter the filename alone.

Full explanations of how to use each program are given below, after the List of commands.

The 11 external commands are:

FORMAT PALETTE DUMP COPY CRC SAMREAD

HIDE UNHIDE PROTECT UPROTECT BATCH

**FORMAT**

This program is used to format a disk to the Pro-Dos standard. This format is the same as the Amstrad PCW 8256 and that machine will read Pro-Dos disks direct and vice versa.

All new disks you wish to use for Pro-Dos files and programs must first be formatted.

FORMAT is issued on it's own and takes no additional arguments.

**A>format**

Would load and run the FORMAT.COM program from drive A.

Once the program has loaded just follow the on-screen prompts. If you have a twin floppy drive SAM you will be given the option of formatting a disk in either drive A or B.

**REMEMBER**

**FORMATTING A DISK DESTROYS ALL DATA THAT MIGHT EXIST ON IT**

Whilst the program is running you are given a graphical display of the tracks and sectors as they are formatted.

Once formatted you will be returned to the system prompt, you can now use the disk for Pro-Dos files.

FORMAT .COM will return to the prompt with one of three messages:

**FORMATTNG COMPLETE**

Formatting operation finished.

**NO DISK IN DRIVE**

FORMAT.COM could not get any drive response.

**DISK IS WRITE PROTECTED**

The write protect tab on the disk is open, Close it and run FORMAT.COM again.

**PALETTE**

This program allows you to alter the screen colours to your own choice.

The Pro-Dos default colours are:

INK = 127 .. Moonlight. (Bright White)
Paper = 27 .. Dusk Blue. (Medium Blue)

A full list of the 128 colours is given on page 66 of the SAM User's Guide.

PALETTE is issued with two numeric arguments separated by a Comma, the first number being the ink value and the second the Paper.

Note: the numbers must be in the range 0 to 127 and the ink and Paper values must be different.

**A>Palette 127,0**

Would load and run the PALETTE.COM program from drive A.

It would change the screen colours to bright white ink on black Paper and return you to the system prompt.

If there are any errors in the command line or if both ink and Paper values are the same, PALETTE .COM will report an error and display the correct syntax to use.

**DUMP**

This program gives you a HEXADECIMAL, and ASCII dump of a named file.

Dump is issued with a single filename argument. Wildcard filenames are not allowed.

**A>dump b:test.com**

Would load and run the DUMP.COM program from drive A.

It would then give you a HEX/ASCII dump of the file TEST.COM which, it would expect to find on drive B.

Once the dump is complete you are returned to the system prompt.

Before running DUMP you can use CTRL-P to obtain a printout of the dumped file as well as screen output, you can also use CTRL-S to invoke screen pauses.

DUMP will return to the system prompt with one of three messages:

FINISHED

DUMP.COM is finished.

FAILED TO OPEN:

DUMP.COM could not find the file you wanted to dump.

AMBGIUOUS FILENAMES NOT ALLOWED

You have used a filename containing Wildcards, this is not allowed with DUMP.COM.

**COPY**

This program allows you to copy files between drives.

Copy is issued with two arguments, the source filename and the destination drive, separated by one or more spaces. The destination drive cannot be the same as the source drive and Wildcards are allowed.

**A>copy *. com b:**

Would Load and run the COPY.COM program from drive A.

It would then copy all the files with the .COM extension from drive A to drive B.

As the files are copied they are listed on screen.

Should COPY.COM find that a file already exists on the destination drive it will pause and give you the option of overwriting it with the "new" copy.

If you want to copy to the current drive (as shown in the system prompt) you do not have to include the second argument when

invoking COPY.COM

**B> a: copy a:**

Would Load and run COPY.COM from drive A.

It would then copy ALL files from drive A to drive B (in this example drive B is the current drive). Note the use of a: in the above example, this is treated as a:*.* (or all files).

COPY.COM will return to the system prompt with one of six messages:

FINISHED

COPY.COM has completed it's operation.

CANNOT COPY TO THE SAM DRIVE

You have tried to copy to the same drive, an example would be:

**A>copy a:**

NO SOURCE FILE FOUND

COPY.COM could not find a named source file.

*** COPY ABORTED ***

An error has occurred and COPY.COM has aborted the operation.

INVALID DESTINATION DRIVE

The destination drive does not exist, an example would be:

**A>copy text.doc c:**

On a single drive SAM, drive C does not exist.

INVALID SOURCE DRIVE

The source drive does not exist, an example would be:

**A>copy  c:text.doc b:**

On a single drive SAM, drive C does not exist.

NOTE: The two examples above would be valid on a twin floppy drive SAM.

**CRC**

This program calculates the checksum of a given file or files.

This can be used to verify if a program is corrupt or has been

tampered with in some way.

CRC is issued with a single filename argument and Wildcards are allowed.

A>crc b: *.com

Would load and run the CRC.COM program from drive A.

It would then calculate and display the checksums of all the files with the .COM extension on drive B.

The filenames are listed to the screen along with the checksum in DECIMAL and HEX (hex shown in brackets).

You can use CTRL-P before invoking CR .COM to get a printout of the filenames and their checksums.

CRC.COM will return to the system prompt with one of three messages:

FINISHED

CRC.COM has completed it’s operation.

NOTHING TO DO

Could not find the named file or files.

FAILED TO OPEN:

An error has occurred in opening the file to check.

**SAMREAD**

There are dozens of good Z80 assemblers and text editors available that run under CP/M 2.2 and thus Pro-Dos.

Using one of these available text editors together with one of the many Macro Z80 assemblers is probably the best way to produce text files, or develop programs for use under Pro-Dos.

That said however, you may wish to use a SAM editor/assembler or a SAM word processor to write programs or text files for use with Pro-Dos.

If this is the case you will have to use SAMREAD to copy the files from the SAMDOS disk to a Pro-Dos formatted disk, or the Pro-Dos RAM disk.

SAMREAD is issued on it’s own and takes no additional arguments.

A>samread

Would load and run the SAMREAD.COM program from drive A.

Once the program has loaded you will be prompted to insert a SAM disk into drive A and press space. You should insert the SAM disk containing the files you wish to copy.

After pressing space SAMREAD will give you a display of all the CODE type files that exist on the SAM disk.

NOTE: SAMREAD will only convert CODE type files it will ignore all other types and will not even display them. If SAMREAD cannot find any CODE type files on the SAM disk you are returned to the system prompt with the following message:

**NO CODE TYPE FILES EXIST ON THAT DISK.**

With the CODE type files displayed on screen you can use the cursor keys to highlight the required file, X to Exit to Pro-Dos and SPACE to select the required file.

After selecting the file you are prompted for the destination drive, this is where you want to copy the file. 0n a single drive SAM you are given the choice of A or B, twin drive SAMS will offer A, B or C.

If you select any drive other than A the file will be copied automatically.

If you selected Drive A then you will need to swap between the Pro-Dos destination disk and the SAM source disk. SAMREAD will prompt you for the required disk as and when it needs it.

Once the copy is complete SAMREAD will report this, and after Pressing SPACE, will re-run, ready to copy another file if required.

NOTE: Because SAMDOS filenames differ from Pro-Dos ones SAMREAD will alter any part of the filename that isn't valid. Therefore, the copied files name may be slightly different from it’s SAMDOS Original.

If ANY disk errors occur, either reading the SAM disk or writing to the Pro-Dos disk SAMREAD will abort.

**HIDE**

This program allows you to set the SYSTEIM flag on a named file or files. With this flag set the file or files remain hidden in a standard disk directory l DIR or DIRN.

However, you can still view files with the system flag set by using DIRS or DIRSN, these commands are explained in chapter 3.

Setting this flag does not affect the usage of the file, it’s only purpose is to hide it.

HIDE is issued with one filename argument, Wildcards are allowed.

**A>hide B: *. com**

Would load and run the HIDE.COM program from drive A.

Once loaded the program will proceed to set the system flag on all the .COM files contained on drive B. As each file is hidden it's name is listed to the screen.

When HIDE.COM has completed it' a operation, it returns to the System prompt. It will also return it an error occurs, the error Will be reported by the system.

HIDE.COM will return with one of the following two messages:

FINISHED

Program completed or an error occurred.

NOTHING TO DO

Could not find the named file or files.

**UNHIDE**

This program is the opposite of HIDE, it's job is to reset the system flag thus making the file or files visible to the DIR and DIRN commands again.

It is used in exactly the same way as HIDE and it's operation and return messages are identical.

Therefore the explanation for HIDE is also applicable to UNHIDE.

**PROTECT**

This program allows you to set the WRITE PROTECT flag on a named file or files. With this flag set the file or files cannot be erased by using the ERA command and no Programs can erase the file or files unless they reset this flag first.

When the ERA command detects that a file has this flag set it will be ignored and therefore not erased.

NOTE: Setting the write protect flag on a file DOES NOT protect it against formatting the disk. The ONLY way to protect against format is to open the write protect tab on the disk itself.

Setting this flag does not affect the usage of the file, it's only purpose is to (software) write protect it.

A file can be both hidden and protected, if you wish.

PROTECT is issued with one filename argument, Wildcards are allowed.

A>protect b: *. com

Would load and run the PROTECT .COM program from drive A.

Once loaded the program will proceed to set the write protect flag on all the .COM files contained on drive B. As each file is protected it's name is listed to the screen.

When PROTECT. COM has completed it's operation, it returns to the system Prompt. It will also return if an error occurs, the error will be reported by the system.

PROTECT.COM will return with one of the following two messages:

FINISHED

Program completed or an error occurred

NOHTNG TO DO

Could not find the named file or files.

**UNPROTECT**

This program is the opposite of PROTECT, it's job is to reset the write protect flag thus making the file or files erasable again.

It is used in exactly the same way as PROTECT and it's operation and return messages are identical.

Therefore the explanation for PROTECT is also applicable to UPROTECT.

**BATCH**

This program allows you to execute a batch of commands as if you were typing them at the command line.

It does this by reading the commands from a batch file.

A batch file is a standard text file saved with the extension .BAT, it must be in pure ASCII format and each Line must be terminated by a carriage return and line feed, therefore, Tasword cannot be used for producing .BAT files.

You are urged to obtain a good CP/M text editor or word processor for producing text files, you can then use it to produce .BAT files , assembler source files Etc .

The Program ZDE.COM comes highly recommended and was the word processor used to produce the original of this user manual!

BATCH is issued with one filename argument, Wildcards are not allowed, this can be (optionally) followed by one or more

variables, each separated by one or more spaces.

The .BAT extension does not have to be specified, BATCH assumes it.

**A>batch test b: source a:**

Would Load and run BATCH.COM

Once loaded BATCH would begin taking command lines from the file TEST.BAT.

b:, source, and a: are three variables to be passed into the .BAT file.

Batch files are used in situations where you have to type a lot of single Line commands, CP/M assemblers are a good example here. A typical assembly session might look like this, (comments are in brackets) and would not be entered on the command Line:

**A> macassem source.abz** (Assemble the file SOURCE to drive B)

**Assembly Complete: 000 Errors.**

**A>B:** (Move to drive B)

**B>a:linkit source.rel** (LINK the file SOURCE .REL)

**Linked File OK. Saved size 512 bytes.**

**B>a:** (Move to drive A)

**A>copy b:source.com** (Copy .COM file to drive A)

As you can see, it involves quite a Lot of typing.

NOTE: The programs used in the above example (MACASSEM and LINKTT) do not actually exist, and are used purely for illustration.

Using a .BAT file would have reduced the above to:

**A> batch abatch source**

This would load and run BATCH .COD which would take command Lines from the file ABATCH. BAT using the variable SOURCE. The actual ,BAT file for the above example ( ABATCH.BAT) would look Like:

```
macassem $1.abz
b:
a:linkit $1.rel
a:
copy b:$1.com
```

ALL the occurrences of $1 will be replaced by variable number one, in the above example the $1 ' s will be replaced with the word

SOURCE.

You can use more than one variable, if your initial command had been:

**A>batch  abatch one two three four five six seven**

You could use the seven variables in the .BAT file with:

**$1, $2, $3, $4, $5, $6 and $7**

Each $number being replaced by it's associated variable. I.E. $7 being replaced with the word Seven.

If you wish to use the $ symbol in one of the .BAT file Lines just type it twice, $$ would be replaced with $.

When BATCH.COM has read in the .BAT file, it executes each line as a command, just as if you had typed it from the command line, replacing the $number symbols with the variables you type after the .BAT filename on the initial command line.

Whilst the .BAT file is being executed you have no control as to what is happening, however, the ESC key will terminate the .BAT file and return control to you.

NOTE: Pressing ESC whilst a program is running from within a .BAT file will not terminate, ESC only terminates before each command in the .BAT file is executed. This may be a little too quick and a single press of ESC is usually not enough. It is best to hold down ESC until control is returned to you.

You can chain .BAT files together by calling BATCH again on the last line of the .BAT file. Here's an example:

```
macassem $1.abz
b:
a:linkit $1.rel
a:copy b:$1.com

batch $2 $3
```

The above is the same .BAT file as before with the exception of the Last line. Note: You can put blank lines between lines in the .BAT file, BATCH.COM strips these out. The above example might be invoked:

**A>batch abatch source nextbat var3**

This would process ABATCH and then re-load BATCH.COM and process NEXTBAT.BAT passing VAR3 into it.

NOTE: VAR3 is variable number 3 (or $3) in the file ABATCH.BAT but when NEXTBAT .BAT is processed it becomes $1 for that file!

BATCH .COM will return to the system prompt when finished, it

returns with no messages. If an error occurs whilst processing the batch file BATCH.COM will return to the system prompt with one of the following seven error messages:

AMBIGUOUS .BAT FILES NOT ALLOWED

BAT filenames cannot contain wildcards.

NAMED EXTENSION MUST BE .BAT

You do not have to name the batch files extension but if you do it must be .BAT.

FAILED TD OPEN OR FIND THE .BAT FILE

Means just that!

.BAT FILE IS TO BIG

The maximum size of a .BAT file is 2048 bytes, or 2k. This is enough space for about 40 commands. If you need more just chain another .BAT file.

EXPANDED LINE  nnnn IS TO LONG

Line number nnnn in the .BAT file is more than 70 characters long. Command Lines can only be 70 characters long, this includes lines in .BAT files. Remember, a .BAT file Line may be less than 70 characters but after any variables have been inserted they may make it longer.

VARIABLE NUMBER ERROR ON LINE nnnn

You have followed a $ symbol with something other than a number or another $ symbol.

A VARIABLE IS OUR OF RANGE ON LINE nnnm

Your .BAT file references a variable that does not exist on the initial command line. I.E. $7 when the command line contains only six variables.

NOTE: when an error occurs and a Line number is given this Line number is the line in the .BAT file with all blank lines removed. E.G.

```
macassem $1.abz

B:

a:linkit $1.rel

A:

Copy b:$1.com
```

As far as BATCH .COM is concerned line number 5 in this example is: copy b:$1 .com

Not: a: linkit $1.rel

As a DOS error occurs during the processing of a .BAT file the command that caused the error is skipped and the next command Processed.

A full list of system (DOS) and command line error messages and their meanings are included in the next chapter.

# Chapter 5

## ERROR MESSAGES

There are three types of possible error message that can occur under Pro-Dos. These are:

COMMAND LINE ERRORS:

These errors occur after entering commands on the command line. Their general cause is bad syntax.

**DOS Error On d:; error message**

Where d is the drive on which the error occurred and error message is the actual error. DOS errors can occur from the command line or press a key, after which the operation that caused the error will be re-tried. There are also DOS errors that will give you the choice of re-trying the operation or quitting. Quitting will cancel the operation that caused the error and if that error occurred from within a program it will then continue with the next operation. Re-trying will attempt to do the operation that caused the error again.

PROGRAM ERRORS:

These errors are reported by a program should it detect something is wrong. These will vary between programs. Sometimes DOS errors will be reported by the system before the error message reported by the program. These may or may not match, if they don't match then ignore the error reported by the program and concentrate on file DOS error, as this is the true definition of the error that actually occurred.

A full list of the Command Line errors and the DOS errors, including their meanings, follows:

**COMMAND LINE ERRORS:**

**INVALID FILENAME**

The filename contains illegal characters or is too long.

**INVALID DRIVE**

You have tried to select a drive that does not exist, or the drive specified in a filename does not exist.

**ONE NUMERIC ARGUMENT ONLY**

You have specified more than one number in the USER command.

**ONE FILENAME ARGUMENT ONLY**

You have specified two or more filenames when the command requires just one.

**INVALID USER NUMBER**

You have followed the USER command with a non numeric symbol or your specified user number is greater than 15.

**INCLUDE USER NUMBER (0 -15)**

You have entered USER without specifying a user number.

**NO FILE**

The file you chose to ERA or Type does not exist.

**NO MORE FILES**

ERA or TYPE with wildcards cannot find any more file matches.

**BAD BLOCK NUMBER, USE: 1 - 245**

You have specified a block number of zero or greater than 245 in the SAVE command.

**FILENAME MUST FOLLOW BL0CK NUIWBER**

You forgot to include the filename after the block number in the SAVE command.

**AMBIGUOUS FILENAME(S) NOT ALLOWED**

You have used a wildcard filename in a command that does not allow it.

**TWO FILENAME ARGUMENTS EXPECTED**

You have forgotten to include the new filename in the REN command.

**TWO FILENAME ARGUMENTS ONLY**

You have included more than two filenames in the REN command.

**NAMED EXTENSTOIS MUST BE COM**

You have tried to load and run a file with an extension other than .COM. Remember, the .COM does not have to be included when you wish to run a program.

**FILE TO BIG**

You on have tried to load and run a .COM file that is too big. Pro-Dos allows .COM files to be a maximum of 62720 bytes (61k).

**CAN'T FIND: name .ext**

The command could not find the file, or the system could not find the .COM file.

**CAN'T EXECUTE FILE IS EMPTY**

You have tried to load and run a .COM file that contains no code. (Just the name is stored on disk.)

**DOS ERRORS:**

NOTE: DOS errors are reported as "DOS Error On d:, " followed by One of the following messages:

**WRITE PRPOTETED**

Pro-Dos could not write to the disk because the write protect tab was open or the software had marked it as write protected. This error also be displayed if a program tries to write to a file that has it's software write protect flag set.

**SEEK FAIL**

Pro-Dos could not read a particular sector on the disk, could be due to the disk not formatted or a corrupt sector.

**BAD CRC**

Pro-Dos could not read a particular sector on the disk because it's checksum did not match the sector contents, normally a corrupt sector.

**NO RESPONSE**

Pro-Dos could not get any drive response, normally because there is no disk in the drive.

**NO DRIVE**

Pro-Dos has been instructed (by a program) to select a drive that does not exist.

**DISK FULL**

There is no more data space left on the disk. Pro-Dos floppy disks can store 706k of data and the RAM disk 124k or 380k depending on your RAM size.

**DIRECTORY FULL**

There is no more space in the disk directory. Pro-Dos floppy disks have a maximum number of 256 directory entries the RAM disk has a maximum of 128.

NOTE: A file may occupy more than one directory entry and as a result the directory may become full but a DIR reveals less than 256 files on the disk.

**FILE EXISTS**

Pro-Dos has been instructed (by a program or the command line) to save a file whose name already exists in the disk directory.

## Appendix

### Appendix Note.

The following appendices give technical information regarding the system as a whole. They are intended as a reference only, and a full understanding of them is not required to use Pro-Dos.

Programmers may find the information contained in them of interest, especially if they intend to write programs to run under Pro-Dos, or indeed, any CP/M 2.2 system.

The information contained in the appendices is written in such a way as to be readable by non-programmers, and as a result is not as detailed as it could be.

Programmers are urged to obtain one of the many technical books available on CP/M 2.2, if they want more detailed descriptions of the systems functions, Etc.

# Appendix

## Appendix A

### Terminal Emulation.

Pro-Dos provide screen escape codes that are compatible with the Popular VT-52 terminal emulator.

You may find that a piece at software is already configured to use this terminal, if not then it will need to be installed.

Most software that requires installation will provide it's own Program to do this, this installation program will almost always include the VT-52 terminal, and this is the terminal you should choose.

If the software does not provide the VT-52 terminal you will have to install it by hand. Most install programs allow this.

All the VT-52 escape sequences consist of the escape character (ASCII value 27 decimal (1B hex)) followed by one or more ASCII alphabetic characters, case IS important in the escape sequences.

I.E. ESC E is not the same as ESC e.

Most install programs will allow you to press the ESC key to signify the escape code, or you may have to type in its hex or decimal value.

The following is a list of the VT-52 escape codes supported by Pro-Dos. Remember, case is important.

The Pro-Dos screen is 80 columns wide by 24 rows deep.

**ESCAPE SEQUENCES.**

**ESC A**

Moves the cursor UP. If the cursor is on the top line it has no effect.

**ESC B**

Moves the cursor DOWN. If the cursor is on the bottom line it has no effect.

**ESC C**

Move the cursor RIGHT. If the cursor is at the right edge of the screen it has no effect.

**ESC D**

Moves the cursor LEFT. If the cursor is at the left edge of the screen it has no effect.

**ESC E**

Clears the screen without affecting the cursor position.

**ESC H**

Homes the cursor to the top left corner of the screen.

**ESC I**

Moves the cursor UP. If the cursor is on the top line the screen is scrolled down.

**ESC J**

Clears the screen from the cursor position to the bottom right hand corner, cursor position is unaffected.

**ESC K**

Erases from the cursor position to the end of the line, cursor position is unaffected.

**ESC L**

Inserts a line. The line containing the cursor and all lines below are scrolled down, the line containing the cursor is then cleared, cursor position is unaffected.

**ESC M**

Deletes a line. The line containing the cursor is deleted and all lines below are scrolled up, the bottom line is then cleared, cursor position is unaffected.

**ESC Y r c**

Moves the cursor to a given position. r is the row number and c is the column . An offset of 32 decimal is added to the values r and c, therefore row 0 column 0 would be expressed as 32, 32, most install programs will prompt you for the offset values.

**ESC d**

Clears the screen from the top left corner to the cursor position, cursor position is unaffected.

**ESC e**

Enables the visible cursor.

**ESC f**

Disables the visible cursor.

**ESC j**

Saves the current cursor position.

**ESC k**

Restores the saved cursor position.

**ESC l**

Clears the line containing the cursor, cursor Position is unaffected.

**ESC o**

Erases from the start of the line to the cursor position, Cursor Position is unaffected.

**ESC p**

Sets INVERSE video mode.

**ESC q**

Sets NORMAL video mode.

# Appendix A

## Appendix B

### Disk Format

As mentioned before, Pro-Dos uses the same basic disk format as the Amstrad PCW 8256.

When obtaining software from a Public Domain library you may have to supply the disk format information, more often than not just stating AMSTRAD PCW 8256 will suffice.

However, the full disk structure and format information is provided below, just in case, and also for reference.

**THE MAIN FLOPPY DISK FORMAT.**

Number of tracks per disk: 160

Number of sides : 2 Alternating.

Lumber of 512 byte sectors per track: 9 (Numbered 1 - 9)

Alternating sides allow the system to view the whole disk as one side with tracks numbered 0 to 159. It does this by the following rule:

| LOGICAL TRACK | SIDE | ACTUAL TRACK |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 0 |
| 2 | 0 | 1 |
| 3 | 1 | 1 ...Etc. |

NOTE: Although the RAM disk is treated like the floppy disks, it does not actually have a format and as a result does not need to be formatted.

**THE DISK PARAMETER BLOCK (DPB).**

Associated with each floppy disk drive and the RAM disk drive is a Disk Parameter Block or DPB.

These provide the system with all it needs to know about how the disk data and directory information is organised.

The DPB concerning the floppy disks is the information that a Public Domain Library may require.

A DPB contains 15 bytes of information divided into 10 fields, some of the field's are 16 bit (or word) values while some are 8 bit (or byte) values. The actual structure of a DPB is given below:

```
+-------------------------------------------------------------+
: SPT : BSH : BLM : EXM : DSM : DRM : AL0 : AL1 : CKS : OFF :
+-------------------------------------------------------------+
  16b    8b    8b    8b    16b   16b   8b    8b   16b   16b
```

The following describes the DPB and it's values for the Pro-Dos floppy disk drives:

| FIELD | MEANING | PRO-DOS VALUE |
|---|---|---|
| SPT ... | Number of 128 byte sectors per track: | 36 |
| BSH ... | Data allocation block shift: | 4 |
| BLM ... | Data allocation block mask: | 15 |
| EXM ... | Extent mask: | 0 |
| DSM ... | Maximum data black number: | 356 |
| DRM ... | Total number of directory entries -1: | 255 |
| AL0 ... | Directory blocks bit mask (high): | 240 |
| AL1 ... | Directory blocks bit mask (low): | 0 |
| CKS ... | Number of checked directory entries/4: | 54 |
| OFF ... | Number of reserved tracks: | 1 |

The following describes the DPB and it’s values for the Pro-Dos RAM disk:

| FIELD | MEANING | PRO-DOS VALUE |
|---|---|---|
| SPT ... | Number of 128 byte sectors per track: | 36 |
| BSH ... | Data allocation block shift: | 4 |
| BLM ... | Data allocation block mask: | 15 |
| EXM ... | Extent mask: | 0 |
| DSM ... | Maximum data block number | 63(191,512k SAM) |
| DRM ... | Total number of directory entries -1: | 127 |
| AL0 ... | Directory blocks bit mask (high): | 192 |
| AL1 ... | Directory blocks bit mask (Low): | 0 |
| CKS ... | Number of checked directory entries/4: | 0 |
| OFF ... | Number of reserved tracks: | 0 |

The use of the words TRACKS and SECTORS are hypothetical when referring to the RAM disk.

The data block size used by Pro-Dos is 2048 bytes (or 2k), hence, the minimum space any file can take on a disk is 2k.

If the above information is meaningless to you, does not worry as Pro-Dos uses it internally,

# Appendix B

## Appendix C

### Memory Usage.

After Pro-Dos has loaded it takes over the whole machine, this means that it does not care what was in the machine when you booted it up, what screen mode you were in or what palette colours you had set.

The first thing Pro-Dos does is to page out the SAM ROM, this is because CP/M (and therefore Pro-Dos) requires all of the 64k address space to be RAM, the ROM remains paged out as Pro-Dos does not use it.

Next it sets the screen mode (mode 3) and palette colours, then figures out how much machine RAM you have (256 or 512k) then looks to see if you have one or two floppy drives t Finally, it configures itself automatically to your particular machines environment.

**MORYORY MAP.**

Pro-Dos makes use of every available RAM byte in both the 256k and 512k SAMS. The following describes the memory map when Pro-Dos is running on a 256k SAM.

Note: TPA stands for TRANSIENT PROGRAM AREA and is the area of RAM that all external programs are loaded to and run from:

| RAM PAGE | | CONTENTS |
|---|---|---|
| 0 | ..... | TPA. |
| 1 | ..... | TPA. |
| 2 | ..... | TPA. |
| 3 | ..... | TPA + System pagers, DPB's and buffers |
| 4 | ..... | System itself. |
| 5 | ..... | 16k Disk cache. |
| 6 to 13 | ..... | RAM Disk. |
| 14 & 15 | ..... | Mode 3 screen + general workspace. |

The following describes the memory map when running on a 512k SAM:

| RAM PAGE | | CONTENTS |
|---|---|---|
| 0 | ..... | TPA. |
| 1 | ..... | TPA. |
| 2 | ..... | TPA. |
| 3 | ..... | TPA + System pagers, DPB's and buffers |
| 4 | ..... | System itself. |
| 5 | ..... | 16k Disk cache. |
| 6 to 29 | ..... | RAM Disk. |
| 30 & 31 | ..... | Mode 3 screen + general workspace. |

The extra 256k of RAM is soaked-up by adding it to the RAM disk.

**SYSTEM PAGE USE.**

When a program is loaded and running Pro-Dos will page itself in and out of the top 16k of address space as and when required.

Pro-Dos pages the screen in and out of the bottom 32k address space as and when required.

NOTE: The "spare" 8k of RAM space (after the 24k mode 3 screen) is used internally by Pro-Dos as general workspace.

When a program (or TPA) is loaded by Pro-Dos it is ALWAYS loaded at address 256 decimal (100 hex), thus following standard CP/M Procedure.

This is also the point at which the Program begins execution and should be your ORG address when writing programs to run under Pro-Dos or, indeed any CP/M system.

The 256 bytes below this address are reserved by the system, therefore no program can exist in this area. This is the same for all CP/Ml systems. Details of this reserved area follow:

**Address 0, 1 & 2 - WARM START ROUTINE.**

This is a JP instruction used when a program has finished running. If you are writing programs to run under Pro-Dos then the correct way to terminate and return to the system is with a RST 0 instruction. You can also terminate with the COLD START System call (see next appendix) or, providing you have not altered the stack pointer, a simple RET as the system restart address is stacked before a program is run. On program entry the stack pointer is at 63492. It is common practice for a program to save the stack pointer, then use it's own stack and restore the stack pointer before the RET. If you choose to terminate by the COLD START system call or a RST 0 then you do not have to restore the stack pointer.

Address 3 - **IOBYTE**

Contains the current Input Output byte , this byte is not used by Pro-Dos and is only included for compatibility.

Address 4 - **CURRENT DEFAULT DRIVE NUMBER.**

0=Drive A, 1=B. . .

Address 5, 6 & 7 - **SYSTEM CALL ROUTINE.**

This is a JP instruction into the system. All programs running under Pro-Dos make system calls via this address. A full list of system functions is included in the next appendix.

Address 8 - 91 - **RESERVED.** System use ONLY.

Address 92 - 127 - **DEFAULT FILE CONTROL BLOCK (FCB).**

Address 128 - 255 - **DEFAULT 128 BYTE DISK I/O BUFFER (DMA).**

All file access under Pro-Dos is carried out using File Control Blocks or FCB' s . All system functions that operate on disk files require DE to point to a valid FCB. An FCB is a 36 byte area containing information such as filename, disk allocation, current record Etc . The structure of an FCB is given below:

```
-----------------------------------------------------------------
:dr:f1:f2://:f8:e1:e2:e3:ex:s1:s2:rc:a0:a1:a2://:an:cr:r0:r1:r2:
-----------------------------------------------------------------
 00 01 02 .. 08 09 10 11 12 13 14 15 16 17 18 .. 31 32 33 34 35
```

Each field is explained below:

dr

Drive code (0-16).

0=Use default drive for file.
1 =Auto select drive A.
2=Auto select drive B. . . . . .

f1-f8

Filename: Uppercase ASCII with bit 7 reset.

e1 -e2

File Extension: Uppercase ASCII with bit 7 reset or set.

Bit 7 of e1 set = WRTTE PROTECT FILE.
Bit 7 of e2 set = SYSTEM file (only seen by a DIRS).
Bit 7 of e3 set = ARCHIVE file, not used by Pro-Dos.

ex

Current file extent. Programmer should set this to zero before Opening a file.

s1- s2

Reserved for system. Programmer must set these to zero before Opening a file.

rc

Record count. Number of 128 byte records in this extent.

a0-an

Disk allocation map. Set by the system after Opening file.

cr

Current record to read or write in a sequential file access. Set to zero to start reading from or writing to the first record in the file.

r0-r2

Random record number (0-65535) with overflow to r3, used when reading or writing randomly. r0 is low byte r1 high.

ALL disk reads and writes take place to and from an area of memory called the DMA. The default DMA is at address 128 but your program can set it to any address (within the allowed space) by using the SET DMA system call.

Disk reads and writes are always in 128 byte records.

On entry to a program Pro-Dos fills the fault FCB address (92) with the filename that followed the program name as typed at the command line. It also falls the allocation map of the default FCB address (108) with the second filename that followed the Program name. It is the Programs job to move this second filename to another area of memory if it is required because an OPEN operation on the first filename will cause the allocation map to overwrite the second filename.

An example, you typed this at the command line:

**TESTPROG b:test.com hello.doc**

On entry to TESTPROG address 92 would hold 2 ( code for drive B), addresses 93 to 103 would hold the ASCII bytes "TEST COM", addresses 104 to 107 hold zero, address 108 holds 0 (code for default drive) and addresses 109 to 119 hold the ASCII bytes "HELLO DOC".

NOTE: Filenames do not contain the. character, they are padded with spaces.

If no filenames are included after the program name then the fields at address 92 and 108 hold ASCII 32 (a space character).

Also, the default DMA (from address 128) holds the complete command line tail following the program name, with the first byte equal to the length of the tail.

In the above example address 128 would hold 21 (21 characters in the command tail) and addresses 129 onwards would hold:

" B:TEST .COM HELLO .DOC"

NOTE: Quotes are not included in the actual command tail and are here to show the exact length of the example tail, also note that the command line is automatically converted to upper case.

Programmers wishing to write programs to run under Pro-Dos (or any CP/M system) are urged to read one of the many books on Programming the system for full information on FCB's and system functions, Etc.

Note: The BREAK button cannot be used in Pro-Dos because there is no guarantee as to what values will be contained at the normal NMI service routine. If you press BREAK whilst in Pro-Dos the machine will nearly always crash.

**YOU HAVE BEEN WARNED!**

If you wish to write programs to run under Pro-Dos you must make sure that they do not use any of the memory above 63493, all the RAM from 63494 to 65535 is reserved by the system.

A tip here, address 6 and 7 hold a 16 bit address, this address -1 is the last byte your program can safely use, this is the same on all CP/M systems and on Pro-Dos this 16 bit address is 63494.

**A NOTE ON INTERRUPTES.**

Pro-Dos uses interrupt mode 2 for its keyboard scanning routines, as a result you should not disable interrupts in any programs you write, or re-vector them elsewhere. When a TPA program is executed the interrupts are enabled and should remain so.

If you disable them you will loose keyboard input.

In the case of re-vectoring them, the system will probably crash because there is no way you can predict the paging that may occur and you could find your interrupt routine paged out!

Remember, CP/M was originally written on a 8080 processor which had no vectored interrupts.

**AGAIN, YOU HAVE BEEN WARNED**!

# Appendix C

## Appendix D

### System Functions.

There are a total of 38, (actually 40 but 2 are reserved) system functions available to programs running under Pro-Dos (and CP/M 2.2), these functions provide the means of communication between program and system.

All the functions are executed by a CALL to address 5.

Any values to pass to a function are passed in the DE pair or E Single registers, the actual system function number is passed in the C register. Any values that the function may return are returned in the accumulator (A register) or the HL pair.

On return from a function, register B equals register H and register A equals register L, all registers (except A, BC and HL) are preserved by the system.

A simple Program to read a single keyboard character and echo it to the screen would be as follows:

```
Org 256                 ;All CP/M programs ORG here
LD C, 1                 ;Function number 1. (Keyboard Input)
CALL 5                  ;Do the function.
RET                     ;After the call register A ( and L) hold
                        ;the character typed.
```

A full list of the 38 functions follow, with entry and exit values described:

---

**FUNCTION 0 ..... RESET SYSTEM.**

ENTRY PARAMETERS . Register C = 0.

EXIT VALUES ..... NIL.

PURPOSE:

Reset SYSTEM returns control to Pro-Dos, this terminates any program that was running and returns you to the command line. RST 0 and JP 0 have the same effect.

---

**FUNCTION 1 ..... KEYBOARD INPUT.**

ENTRY PARAFIETERS . Register C = 1 .

EXIT VALUES ..... Register A = ASCII character typed .

Purpose:

KEYBOARD INPUT returns the ASCII value of the next character type at the keyboard, the function does not return until a character has been typed. The character is automatically echoed to the screen (and the printer if CTRL-P is ON).

---

**FUNCTION 2 ..... TERMINAL OUTPUT.**

ENTRY PARAMETERS . Register C = 2.
E = ASCII character to display .

EXIT VALUES ..... NIL.

PURPOSE:

TERMINAL OUTPUT displays the ASCII character contained register E, it is also sent to the printer if CTRL-P is ON.

---

**FUNCTION 3 ..... ALTERNATIVE INPUT.**

ENTRY PARAMETERS . Register C = 3.

EXIT VALUES ..... Register A = ASCII character typed.

PURPOSE:

ALTERNATIVE INPUT provides an alternative means of character input. This function is not used and is treated as a KEYBOARD INPUT (function 1).

---

**FUNCTION 4 ..... ALTERNATIVE OUTPUT.**

ENTRY PARAMETERS . Register C = 4.
E = ASCII character.

EXIT VALUES ..... NIL.

PURPOSE:

ALTERNATIVE OUTPUT provides an alternative means of character output. This function is not used and is treated as a PRINTER OUTPUT (function 5).

---

**FUNCTION 5 ..... PRINTER OUTPUT.**

ENTRY PARAMETERS . Register C = 5.
E = ASCII character .

EXIT VALUES ..... NIL.

PURPOSE:

PRINTER Output sends the ASCII character contained in the E register to the Centronics port of the SAM COMMS interface.

---

**FUNCTION 6 ..... DIRECT INPUT/OUTPUT.**

ENTRY PARAMETERS . Register C = 6.
E = 255 for input or character to output.

EXIT VALUES ..... Register A = Keyboard status or inputted character.

PURPOSE:

DIRECT INPUT / OUTPUT allows a program to check for keyboard characters without the system waiting for 3 key press, it also allows terminal output. If register E contains 255 on entry then the function will return A = 0 for no available keyboard character, or A = ASCII keyboard character, if one was available. If register E <> 255 on entry , then the ASCII character contained in E is displayed on screen, and also sent to the printer (if CTRL-p is ON) .

---

**FUNCTION 7 ..... GET I/O BYTE.**

ENTRY PARAMETERS . Register C = 7.

EXTT VALUES ..... Register A = I/O BYTE value.

PURPOSE:

GET I /O BYTE will return with the current value of the I/O BYTE (stored at address 3). This byte serves no Purpose to Pro-Dos, however, some programs may use it.

---

**FUNCTION 8 ..... SET I/O BYTE.**

ENTRY PARAMETERS . Register C = 8.
E = I/O BYTE value.

EXTT VALUES ..... NIL.

PURPOSE:

SET I/O BYTE will set the I/O BYTE to the value contained in register E.

---

**FUNCTION 9 ..... STRING OUTPUT.**

ENTRY PARAMETERS . Register C = 9.
DE= String Address.

EXIT VALUES ..... NIL.

PURPOSE:

STRING OUTPUT will display the text string stored at address DE until a $ symbol is encountered in the string, it is also sent to the printer if CTRL-P is ON.

---

**FUNCTION 10 ..... TYPE INTO BUFFER.**

ENTRY PARAMETERS . Register C = 10.
DE= Buffer Address.

EXIT VALUES ..... Buffer address filled with typed characters.

PURPOSE :

TYPE INTO BUFFER allows input of a string of keyboard characters, it is similar to the command Line, and all the editing features available on the command line are available to this function. Register pair DE points to the address of the buffer to read characters to, the program must load the first byte of this buffer with the maximum number of allowed characters (1 to 255). When the function returns the second byte of the buffer address will hold the number of characters that were typed and the characters themselves follow this, from buffer address +3.

---

**FUNCTION 11 ..... GET KEYBOARD STATUS.**

ENTRY PARAMETERS . Register C = 11.

EXIT VALUES ..... Register A = Keyboard status.

PURPOSE:

GET KEYBOARD STATUS will return with register A holding 255 if a keyboard character is available, else A = 0.

---

**FUNCTION 12 ..... VERSION NUMBER.**

ENTRY PARAMETERS . Register C = 1 2 .

EXIT VALUES ..... Register A = Version Number.

PURPOSE:

VERSION NUMBER returns with the CP/M version number in register A, this is a hexadecimal value, Pro-Dos returns 22 hex to signify Version 2.2.

---

**FUNCTION 13 ..... DISK RESET.**

ENTRY PARAMETERS . Register C = 13.

EXIT VALUES ..... NIL.

PURPOSE:

DISK RESET will reset all drive buffers including any software write Protection, it also resets the default DMA buffer to address 128. Drive A is then selected.

---

**FUNCTION 14 ..... DISK SELECT.**

ENTRY PARAMETERS . Register C = 14.
E = Disk Code Number to Select.

EXIT VALUES ..... Register HL= 0 if the drive code is invalid

PURPOSE :

DISK SELECT makes the drive named in register E into the current default drive for all further file access. Drive code = 0 for drive A, 1 for B, Etc.

---

**FUNCTION 15 ..... OPEN A FILE.**

ENTRY PARAMETERS . Register C = 1 5.
DE= FCB address.

EXIT VALUES ..... Register A = Directory code.

PURPOSE:

OPEN A FILE will scan the disk directory referenced in the first byte of the FCB, (pointed to by DE) for the filename contained in bytes 1 to 1 2. If it finds the filename, then the disk allocation information is copied to bytes 15 to 31 and register A equals the value 0 , 1 , 2 or 3 to indicate a successful OPEN, else register A equals 255 meaning the file could not be found. Pro-Dos imposes no Limitations on how many files can be open. Before a file can be read or written to it must be opened and if reading or writing is to start from the first record then byte 32 (cr) of the FCB must be set to zero. An ASCII ? in any position

in the FCB filename will match any character in that position in the directory filenames.

---

**FUNCTION 16 ..... CLOSE A FILE.**

ENTRY PARAMETERS . Register C = 16.
DE= FCB address

EXIT VALUES ..... Register A = Directory code.

PURPOSE:

CLOSE A FILE performs the opposite of OPEN A FILE, it is used to permanently store any updated FCB information on to the disk. The exit and entry values of OPEN A FILE are applicable to CLOSE A FILE. If an OPEN file has been written to then it must be closed to ensure the file information on disk is up to date. If, however, only read operations took place on the OPEN file then you do not have to close it.

---

**FUNCTION 17 ..... FIND FIRST.**

ENTRY PARAMETERS . Register C = 17.
DE= TCB address.

EXIT VALUES ..... Register A = Directory code.

PURPOSE:

FTND FIRST will search the disk referenced by the first byte of the FCB for the filename contained in byte 1 to 12. If the file cannot be found then FIND FIRST will return with register A containing 255. If the file was found then the current DMA address is filled with the 128 byte directory record that contains the file, A is returned with a value from 0 to 3. You can multiply register A by 32 and add it to the DMA address to point to the matched filename. An ASCII ? in the FCB filename will match any character at that position in the disk directory filenames.

---

**FUNCTION 18 ..... FIND NEXT.**

ENTRY PARAMETERS . Register C = 18.

EXIT VALUES ..... Register A = Directory code.

PURPOSE:

FIND NEXT will continue with the disk directory scan from the last matched filename as returned by FIND FIRST. Exit values and

conditions for FIND FIRST also apply to FIND NEXT.

---

**FUNCTION 19 ..... ERASE FILE.**

ENTRY PARAMETERS . Register C = 19.
DE= FCB address.

EXIT VALUES ..... Register A = Directory code.

PURPOSE:

ERASE FILE will erase all filenames that match the FCB addressed by DE (which may contain ? characters). ERASE FILE returns with register A containing 255 if no file or files could be found otherwise a value from 0 to 3 is returned.

---

**FUNCTION 20 ..... SEQUENTIAL READ.**

ENTRY PARAMETERS . Register C = 20.
DE= FCB address.

EXIT VALUES ..... Register A = Directory code.

PURPOSE:

SEQUENTIAL READ will read the next 128 byte record to the current DMA buffer address. The record is contained in the "CR" byte of the FCB. The FCB must have been activated with an Open or MAKE operation before reading can take place. After the current record has been read the CR byte is incremented and if it overflows, the next logical file extent is automatically Opened and the CR byte reset to zero. This function returns with register A holding zero if the read was successful else register A contains a non-zero value do indicate END OF FILE.

---

**FUNCTION 21 ..... SEQUENTIAL WRITE.**

ENTRY PARAMETERS . Register C = 21.
DE= FCB address.

EXIT VALUES ..... Register A = Directory code.

PURPOSE:

SEQUENTIAL WRITE will write the next 128 byte record from the current DMA buffer address. The record is contained in the "CR" byte of the FCB. The FCB must have been activated with an OPEN or MAKE operation before writing can take place. After the current record has been written the CR byte is incremented and if it overflows the next logical file extent is automatically OPENed

and the CR byte reset to zero. This function returns with register A holding zero if the write was successful else register A contains a non-zero value to indicate a write error. Note: You can write into an existing file and all new records written will overlay the old ones.

---

**FUNCTION 22 ..... MAKE A FILE.**

ENTRY PARAMETERS . Register C = 22.
DE= FCB address.

EXIT VALUES ..... Register A = Directory code.

PURPOSE:

MAKE A FILE will construct a new file on disk, the FCB pointed to by DE must not contain any ? characters and the filename it contains must not already exist on disk. MAKE A FILE also has the effect of OPENing it too, so an OPEN call is not needed after a MAKE. MAKE A FILE returns with register a holding 255 to indicate an error in making the file else register A holds a value from 0 to 3.

---

**FUNCTION 23 ..... RENAME.**

ENTRY PARAMETERS Register C = 23.
DE= FCB address.

EXIT VALUES ..... Register A = Directory code.

PURPOSE:

RENALWE will change the filename referenced in the first 16 bytes at the FCB into the filename contained in the second 16 bytes. Neither filenames may contain ? characters. RENAME returns with A holding 255 if the first filename could not be found else register A holds a value from 0 to 3.

---

**FUNCTION 24 ..... DRIVES LOGGED.**

ENTRY PARAMETERS . Register C = 24.

EXIT VALUES ..... Register HL = Drives logged in.

PURPOSE:

DRIVES LOGGED returns the HL pair holding the drives that have been used by the system. HL is a 16 bit value with bit 0 of L equal to drive A and bit 7 of H equal to drive P. A zero bit indicates a drive that has not been used while a non-zero bit

indicates a drive that has been used.

---

**FUNCTION 25 ..... CURRENT DISK.**

ENTRY PARAMETERS . Register C = 25.

EXIT VALUES ..... Register A = Current Disk.

PURPOSE:

CURRENT DISK returns with register A containing the code for the currently selected disk. Code 0 = Drive A, 1 = B, Etc.

---

**FUNCTION 26 ..... SET DMA BUFFER ADDRESS.**

ENTRY PARAMETER . Register C = 26.
DE= DMA address.

EXIT VALUE ..... NIL.

PURPOSE:

SET DMA BUFFER ADDRESS will set the DMA buffer for disk read and writes to the address in register pair DE. This address will then remain until another SET DMA ADDRESS, RESET SYSTEM or DISK RESET function occurs.

---

**FUNCTION 27 ..... GET ALLOCATION.**

ENTRY PARAMETERS . Register C = 27.

EXIT VALUES ..... Register HL= Allocation map address.

PURPOSE:

GET ALLOCATTON will return with HL Pointing to the disk allocation map for the current drive.

---

**FUNCTION 28 ..... SOFTWARE WRITE PROTECT.**

ENTRY PARAMETERS . Register C = 28.

EXIT VALUES ..... NIL.

PURPOSE:

SOFTWARE WRITE PROTECT will mark the current drive as write protected, the system will then treat any disk in this drive as



if the write protect tab were OPEN. Note: This does not protect against FORMATTING the disk.

---

**FUNCTION 29 ..... GET WRITE PROTECTED.**

ENTRY PARAMETERS . Register C = 29.

EXIT VALUES ..... Register HL= Drives write Protected.

PURPOSE:

GET WRITE PROTECTED returns the HL pair holding the drives that have been marked as software write protected. HL is a 16 bit value with bit 0 of L equal to drive A and bit 7 of H equal to drive P. A zero bit indicates a drive that can be written to, while a non-zero bit indicates a drive that is protected.

---

**FUNCTION 30 ..... SET FILE FLAGS.**

ENTRY PARAMETERS . Register C = 30.
DE= FCB address.

EXIT VALUES ..... Register A = Directory code.

PURPOSE:

SET FILE FLAGS allows you to set or reset the system, archive and read Only flags on the file referenced by the FCB pointed to by DE. The FCB cannot contain ? characters and it is the programs responsibility to set or reset bit 7 on each of the three filename extension characters . SET FILE FLAGS returns with register A holding 255 for no file found else register A hold the value 0 to 3.

---

**FUNCTION 31 ..... GET DPB.**

ENTRY PARAMETERS . Register C = 31.

EXIT VALUES ..... Register HL = DPB address.

PURPOSE:

GET DPB will return with HL pointing to the disk parameter block for the current drive.

---

**FUNCTION 32 ..... GET/SET USER NUMBER.**

ENTRY PARAMETERS . Register C = 32.

E = 255 to get, 0 to 15 to set

EXIT VALUES ..... Register A = Current user number if E = 255.

PURPOSE:

GET/SET USER NUMBER will return with register A holding the current user number, (0 to 15) if the function was called with register E holding 255, else the function will set the user number to that contained in register E.

---

**FUNCTION 33 ..... RANDOM READ.**

ENTRY PARAMETERS . Register C = 33.
DE= FCB address.

EXIT VALUES ..... Register A = Error code.

PURPOSE:

RANDOM READ is similar to the SEQUENTIAL READ operation, except the record number is treated as a 24 bit value and is stored in Locations 33, 34 and 35 of the FCB. Location 33 is the low byte 34 is the high byte and 35 is the overflow byte. The overflow byte must always contain zero, a non-zero value indicate an overflow past the end of the file. Locations 33 and 34 of the FCB are treated as a 16 bit value which is equal to the record you wish to read. File extents for the required record are all worked out automatically by the system and the record is read to the current DMA address. Unlike SEQUENTIAL READ the record number is not increased, therefore, subsequent random reads will read the same record. Random read will return with register A holding one of the following codes:

0 = Operation OK.
1 = Reading unwritten data.
3 = Cannot close current extent
4 = Seek to unwritten extent.
6 = Seek past end of disk.

---

**FUNCTION 34 ..... RANDOM WRITE.**

ENTRY PARAMETERS . Register C = 34.
DE= FCB address.

EXIT VALUES ..... Register A = Error code.

PURPOSE:

RANDOM WRITE is similar to the Sequential WRITE operation, except the record number is treated as a 24 bit value and is stored in

locations 33, 34 and 35 of the FCB. Location 33 is the low byte, 34 is the high byte and 35 is the overflow byte. The overflow byte must a-ways contain zero, a non-zero value indicates an overflow past the end of the file. Locations 33 mid 34 of the FCB are treated as a 16 bit value which is equal to the record you wish to write. File extents for the required record are all worked out automatically by the system and the record is written from the current DMA address. Unlike SEQUENTTAL WRITE the record number is not increased, therefore, subsequent random writes will write the same record. RANDOM WRITE returns the same error codes as RANDOM READ with the exception of the following:

5 = Couldn't create new extent.

---

**FUNCTION 35 ..... CALCULATE FILE SIZE.**

ENTRY PARAMETERS . Register C = 35.
DE= FCB address.

EXTT VALUES ..... FCB locations 33, 34 and 35 set by the function.

PURPOSE:

CALCULATE FILE SIZE will scan the disk directory for the file addressed by the FCB pointed to by the register pair DE, the FCB filename cannot contain any ? characters. If the file is found the function will set the RANDOM RECORD NUMBER (Locations 33, 34 and 35 of the FCB) to the value of the last record in the file + 1. If the file contains the maximum amount of records (65536) then location 35 of the FCB will be set to 1.

---

**FUNCTION 36 ..... SET RANDOM RECORD NUMBER.**

ENTRY PARAMETERS . Register C = 36.
DE= FCB address.

EXIT VALUES ..... FCB locations 33, 34 and 35 set by the function.

PURPOSE:

SET RANDOM RECORD NUMBER will set the FCB Locations 33 , 34 and 35 to the current random record number for the file addressed by the FCB, pointed to by the DE register pair . This function is used to switch from a SEQUENTIAL READ or WRITE to a RANDOM READ or WRITE.

---

**FUNCTION 37 ..... RESET CHOSEN DRIVES.**

ENTRY PARAMETERS . Register C = 37.
BE= Drive vector.

EXIT VALUES ..... Register A = 0.

PURPOSE:

RESET CHOSEN DRIVES will reset the drives specified in the DE register pair. Bit 0 of register E is drive A whilst bit 7 of register D is drive P.

---

**FUNCTION 38 ..... RESERVED FUNCTION.**

---

**FUNCTION 39 ..... RESERVED EUNCTIQN.**

---

**FUNCTION 40 ..... RANDOM WRITE WITH ZERO FILL.**

ENTRY PARAMETERS . Register C = 40.
DE= FCB address.

EXIT VALUES ..... Same as RANDOM WRITE Function.

PURPOSE

RANDOM WRITE WITH ZERO FILL is similar to the RANDOM WRITE function except that any unallocated data blocks are filled with zeros before the data is written.

## USER MANUAL AMENDMENTS

[*The amendments information came to me in Jan 2005*]

At the time of printing, the following information was omitted from the user manual.

---

**ERROR MESSAGES**

The DOS error: WRITE PROTECTED, described on page 37 of the user manual should have included the following:

"....  it as write protected.  This error will also be displayed if a program tries to write to a file that has it's software write protect flag set."

---

**KEY RETURN VALUES**

When installing software you may have to specify certain key return codes, most commonly the cursor and delete keys. Pro-Dos returns the following key codes, shown in decimal and hex:

| KEY | RETURN | VALUE |
|---|---|---|
| DELETE (un-shifted) | 127 | (7F) |
| DELETE (shifted | 8 | (08) |
| ESC | 27 | (1B) |
| TAB | 9 | (09) |
| CURSOR UP | 31 | (1F) |
| CURSOR DOWN | 30 | (1E) |
| CURSOR RIGHT | 29 | (1D) |
| CURSOR LEFT | 28 | (1C) |

All other keys return their equivalent ASCII values.

---

**FUNCTION KEYS**

The function keypad (F0 - F9), un-shifted, has been set to return the popular Wordstar (tm) cursor control keys. These are as follows:

| FUNCTION KEY | RETURN VALUE | WORDSTAR FUNCTION |
|---|---|---|
| F0 | CTRL-Z | SCROLL DOWN |
| F1 | CTRL-R | PAGE UP |
| F2 | CTRL-X | CURSOR DOWN |
| F3 | CTRL-C | PAGE DOWN |
| F4 | CTRL-S | CURSOR LEFT |
| F5 | CTRL-W | SCROLL UP |
| F6 | CTRL-D | CURSOR RIGHT |
| F7 | CTRL-A | WORD LEFT |
| F8 | CTRL-E | CURSOR UP |
| F9 | CTRL-F | WORD RIGHT |

Function keys F0 - F9, shifted, will return the numbers 0 - 9.

Many pieces of software are already configured to use the Wordstar (tm) cursor control functions.

End of user manual amendments.

---