DOS/65 SYSTEM DESCRIPTION

VERSION 2.1

TABLE OF CONTENTS

SECTION 1 - INTRODUCTION

Of all the problems which plague the 6502 enthusiast, the lack of a low-cost, compatible operating system is one of the most critical. The operating system, the media, and the floppy disk hardware of the common 6502 systems (Commodore, Apple, OSI, Atari, and AIM/SYM/KIM) are not compatible and can not readily be interfaced to systems other than the original. This incompatibility has been the major factor which has prevented the 6502 machines from achieving the status that other systems have achieved.

The challenge then is to establish a standard which minimizes cost, allows easy use by different systems, and ultimately will help make the 6502 and its derivatives the universal processor it is capable of being.

What I have done is attack the software side of the problem in order to make any 6502 system a truly workable disk based system. In addition a degree of compatibility is now possible not only between 6502 systems but with large parts of the world of CP/M systems. The result of my efforts is a system of software which I have named DOS/65.

Like many previous efforts, DOS/65 initially relied upon hardware from the IEEE 696 (S-100) bus and is still available for some IEEE 696 hardware. One can also build your own controller as many DOS/65 users have done and standard OSI and Commodore disk drives are supported. DOS/65 is not limited to any specific hardware platform.

The hardware interface between the 6502 system and the IEEE 696 bus is discussed in detail in the IEEE 696 INTERFACE GUIDE that is available from Richard A. Leary. That interface is not complex and even in the case of most disk controllers involves little more than would be required to use IEEE 696 memory or I/O boards with the 6502.

DOS/65 itself is a software system which has the same basic structure as the foremost 8080/Z80 software standard, CP/M, and is file compatible with CP/M Version 1.4 and later versions. Since the machine language software obviously can not be compatible between the 8080/Z80 and the 6502, the term "file compatible" simply means that a DOS/65 diskette could be inserted into a CP/M system or vice-versa and the files manipulated in every way except actual execution.

The system could not tell the difference between files created on the host versus those created on a CP/M system. For example a BASIC-E source program which had been created, edited, and used on a CP/M system could be used on a DOS/65 system using the BASIC-E/65 compiler and interpreter with at most only minor changes. Data files of all forms could be exchanged between systems without restriction.

The reasons for selecting CP/M as the standard are simple. CP/M has many features including:

- Nearly total hardware independence
- Easy access to operating system primitives
- Easy alteration to accommodate system unique characteristics
- Large library of compatible software that in many cases can be ported to DOS/65
- Simplicity

None of these claims are overwhelming in themselves, but taken as a whole they stand as a compelling reason to use CP/M as the format and structure standard for a 6502 operating system.

The remainder of this document is devoted to a description of DOS/65 and an explanation of how to use DOS/65.

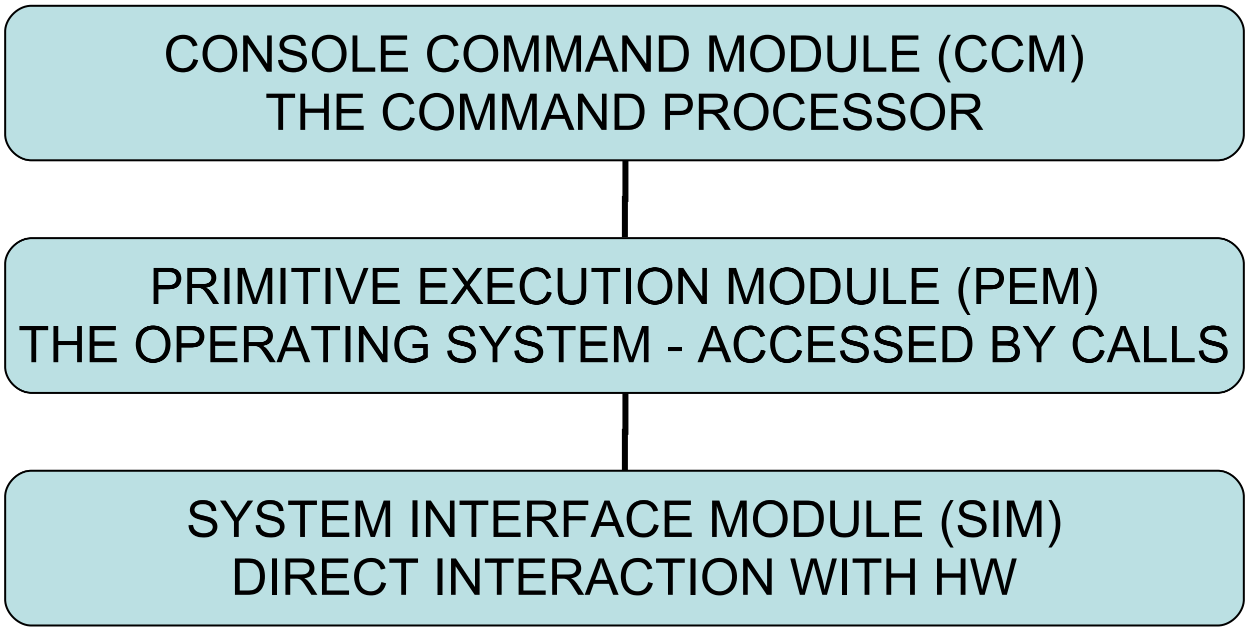
SECTION 2 - SOFTWARE AND OPERATIONS

2.1 OVERVIEW

The following material discusses use of the system. It assumes that the user has successfully booted the system and is prepared to begin using the system. If that is not the case, please consult the appropriate manuals.

2.2 STRUCTURE

DOS/65 is a modular system organized in layers as shown in the following figure. In DOS/65 the layers are named:


CONSOLE COMMAND MODULE (CCM)
THE COMMAND PROCESSOR
PRIMITIVE EXECUTION MODULE (PEM)
THE OPERATING SYSTEM - ACCESSED BY CALLS
SYSTEM INTERFACE MODULE (SIM)
DIRECT INTERACTION WITH HW

As the figure indicates, the layers of DOS/65 are strongly related to both the degree of host independence and the ease of use of that layer by the programmer. Each layer is described below.

2.2.1 CCM

CCM is the primary interactive interface between DOS/65 and the user. It is through the "plain-language", hardware independent media of CCM that the user can request execution of the built-in high level commands, can alter the default drive, or can execute a transient program such as BASIC-E/65. In the Version 2.1 of DOS/65 the available CCM command formats are:

```
DIR (drive:)(object)

SAVE (length) (drive:)(ufn) (address)

ERA (drive:)(object)
```

```
GO (address)

LOAD (drive:) (ufn) (address)

TYPE (drive:)(ufn)

REN (drive:)(ufn)=(drive:)(ufn)

(drive:)

(drive:)(ufn)
```

2.2.1.1 PARAMETERS

2.2.1.1.1 OBJECT/UFN/AFN

The (object) field is a file specification identical to the CP/M file specification convention. It can either be an unambiguous file name (UFN) or an ambiguous file name (AFN). An UFN consists of a one to eight character file NAME optionally separated from a zero to three character TYPE field by a ".". The following are examples of valid UFN's:

```
BASIC.COM
TEST
.A
A$TEST.$$$
```

The characters

`. ? * = : < > ;` and `DELETE($7F)`

are illegal as are all lowercase alphabetic characters. The restriction against lowercase characters does not cause problems when used with a console having lowercase characters since CCM translates all console inputs to uppercase.

<u>CAUTION</u>

Lowercase to uppercase translation is only done when operating in CCM. No translation is performed when operating in a transient program unless otherwise specified.

The following are not legal UFN's:

```
BSC..TST
TEST<A.SRC
AB:
```

An AFN follows the same rules as an UFN except that the wildcard characters `*` and `?` are used. Use of `?` means that the specified AFN will match any file which has any character in the corresponding character position. Thus the AFN `BA?.COM` would match the files `BAS.COM` and `BA$.COM` but not the file `BBS.COM`. The `*` simply causes that character and all following characters in the associated field to be the same as `?`. Thus `B*.COM` is equivalent to `B???????.COM`.

That UFN would match

`BAS.COM`, `BA$.COM`, and `BBS.COM`.

<u>NOTE</u>

Remember that `*.*` matches every file on the diskette.

2.2.1.1.2 DRIVE

DOS/65 can handle up to eight drives. Each drive is designated by a letter (A, B, C, ..., or H) and while operating in CCM all command input prompts are of the form `x>` where `x` is the letter corresponding to the currently active (default) drive.

DOS/65 allows an AFN or UFN to be preceded with a drive specification consisting of the drive letter followed by a colon. If none is specified, the default drive is assumed for all file references. For example, a command such as

```
DIR B:
```

will list the directory of drive B: regardless of the current default drive.

2.2.1.1.3 LENGTH

The LENGTH parameter is an integer numerical field from 0 to 255. If entered as a simple number it is assumed to be a decimal number. If preceded by a $, the field is evaluated as a hexadecimal number. The evaluated number represents the number of 256 byte pages to be manipulated.

2.2.1.1.4 ADDRESS

The ADDRESS parameter is an integer numeric field from 0 to 65535. If entered as a simple number it is assumed to be a decimal number. If preceded by a $, the field is evaluated as a hexadecimal number. The evaluated number represents a 16-bit address within the 6502 address space.

2.2.1.2 CCM INPUT AND INPUT EDITING

Command lines input to CCM are buffered as a full line and can be edited prior to execution. The following keys have special meaning to CCM.

<u>KEY</u> <u>ACTION</u>

(ctl-x)	Cancels current input line and lets user enter new command line.
(ctl-i)	Enters a tab (ctl-i) into the buffer and spaces the console to the next modulo eight column.
(ctl-r)	Types the current buffer contents onto the console.
(delete)	Deletes the last character input by the user.
(ctl-c)	If the first input in a line will cause a WARM BOOT to be executed.
(cr)	Terminates input and causes execution of the command currently in the buffer.
(ctl-p)	Toggles the flag which causes all output to the console to also be output to the "LIST" device.

2.2.1.3 COMMANDS
Now that we know what the range of valid file designators is, let's define what the built-in commands do.

2.2.1.3.1 DIR (drive:)(object)
Displays a listing on the console of the disk directory for all files matching the specified file designator. A blank object field is equivalent to `*.*`. If no files are found which match the specification, the user is so informed. Examples:

`dir *.asm`	show assembler files on default drive
`dir b:`	show all files on drive b

2.2.1.3.2 SAVE (length) (drive:)(ufn) (address)
Transfers the contents of (length) decimal pages from memory to the file (ufn). If a file already exists named (ufn), it is erased. If no address parameter is entered, the transfer begins at the TEA start address for the host system and can include the full TEA. If the address parameter is entered, the transfer begins at the specified address. Examples:

`save 15 b:*xas.com`	save 15 pages on b as *xas.com
`save 1 .com $400`	save 1 page on default as .com starting at hexadecimal 400

2.2.1.3.3 ERA (drive:)(object)
Erases all files matching the file designator. ERA *.* will result in request for verification that the user wishes to erase all files on the disk. Examples:

era b:*.com	erase all com files on drive b
era test.asm	erase file test.asm on default

2.2.1.3.4 GO (address)
Transfers control to the instruction at the (address) entered. If no (address) is entered, then program execution begins at the TEA start. Examples:

go	start execution at TEA
go $2000	start execution at hexadecimal 2000 regardless of TEA start

2.2.1.3.5 LOAD (drive:)(ufn) (address)
Loads the specified file as a binary file at the address stated. If no address is specified, the load will start at the TEA start. In all cases, the file is loaded as a binary file regardless of the file contents or name. Examples:

load test.bin	load test.bin at TEA
load newsys.kim $800	load newsys.kim at hexadecimal 800 regardless of TEA start

2.2.1.3.6 TYPE (drive:)(ufn)
Transfers the designated file (hopefully an ASCII text file) to the console. Horizontal tabs (ctl-i or $09) will be expanded on 8 column spacing. Transfer will be terminated by a physical end of file or by the DOS/65 EOF character (ctl-z or $1A). Typing a (ctl-s) during execution will freeze the output as discussed in Section 2.2.2 of the SYSTEM INTERFACE GUIDE. Any other character will cause the TYPE function to be terminated. Examples:

type b:sim.asm	type file sim.asm from drive b
type report.bas	type file report.bas from default

2.2.1.3.7 REN (drive:)(ufn) (drive:)(ufn)
Renames the file which matches the first designator to the second designator. If a file already exists matching the second designator, the command will not be executed and an error message will be displayed. Examples:

ren report.old report.bak	rename report.old on default drive to report.bak
ren b:data b:data.com	rename data to data.com on drive b

2.2.1.3.8 (drive:)
Changes the default drive to the specified drive. Examples:

b:	default drive will be B
a:	default drive will be A

2.2.1.3.9 (drive:)(ufn)
The last capability of CCM is the ability to load and execute transient (i.e., not "built-in") commands. That is done by entering the NAME portion only of a UFN of type "COM". CCM searches for the file of type "COM" having the NAME corresponding to the command line input. If it is found, it is loaded and executed. For example, if a BASIC interpreter existed in binary code form as the file BASIC.COM on drive B, entering the CCM command

```
b:basic
```

causes that file to be loaded into RAM from drive B and executed. The load start and execution entry address is the same as the SAVE start address discussed in section 2.2.1.3.2. That address is called the TRANSIENT EXECUTION AREA (TEA). The following table shows the values of TEA START for the various DOS/65 configurations.

CONFIGURATION	TEA START
S	$200
P	$400
A	$800
T	$1000
R	$1400
K	$2000

The CCM for each configuration is slightly different since it must know where TEA starts for GO, LOAD, and SAVE and transients to work correctly. Obviously COM files for one version can not be directly executed on another configuration. There are ways around that limitation which allow the user to load and execute files having a TEA higher than the current configuration by using the DOS/65 standard transient DEBUG. Consult the DEBUG manual for further details.

In addition to finding and loading the transient program, CCM will attempt to build up to two FCB's from the characters following the transient UFN on the command line. Regardless of whether CCM can build any FCB's from the command line, CCM will transfer all characters after the transient UFN to a command line buffer at the DEFAULT BUFFER location. Additional data on both these two features is included in Appendix C.

Note that CCM is in one sense only a somewhat special transient program. While it is not loaded and executed at TEA the way a transient is, it is like a transient in some ways. For example, no transient uses CCM and hence the transient may use the region normally occupied by CCM without any ill effects as long as a WARM BOOT is executed upon exit from the transient. CCM can not be overlaid during load of the transient by CCM.

The major transient programs supplied with DOS/65 are described in individual manuals. For those programs which do not have a separate manual their use is described in Appendix A.

2.2.2 PEM

PEM is the core of the system. It is PEM which user developed programs usually must interface with in order to execute the DOS/65 console, peripheral, and disk primitives. PEM executes a function based upon a function number passed to it in the X register and data or address information passed to it in the A or A and Y registers. The command summary shown in Table 2-1 is probably totally confusing at this point (especially for the disk commands) since a great deal of information is packed into the table and a lot of other knowledge is assumed. Full details on each command are contained in the SYSTEM INTERFACE GUIDE.

2.2.2.1 DISK FILE CONCEPTS

For the disk commands the underlying assumptions upon which PEM operates are:

1. For each file PEM maintains one or more directory entries. Each directory entry is an "extent" and, if full, refers to 16K (K=1024) bytes of disk storage. Under some conditions directory entries can actually refer to more than 16K bytes.

2. Each directory entry is organized as sixteen 1K, eight 2K, four 4K, two 8K, or one 16K byte blocks. A block is always assigned as an entity to a given file even if only part of it is actually used.

3. The blocks assigned to each extent are ordered in a logically but not necessarily physically contiguous manner. Thus an extent having two blocks assigned (such as $07 and $13) is a continuous file whose actual physical location on the disk is determined by the block number and the mapping scheme used by PEM. That scheme is totally transparent to the user.

TABLE 2-1. PEM COMMAND SUMMARY

Command Number	**Function**
0	WARM BOOT
1	READ CONSOLE INPUT WITH ECHO
2	CONSOLE OUTPUT
3	READ FROM READER
4	WRITE TO PUNCH
5	WRITE TO LIST DEVICE
6	READ CONSOLE INPUT W/O ECHO
7	READ I/O STATUS
8	SET I/O STATUS
9	PRINT BUFFER
10	READ BUFFER
11	CONSOLE READY
12	READ LIST STATUS
13	INITIALIZE SYSTEM
14	SELECT DRIVE
15	OPEN FILE
16	CLOSE FILE
17	SEARCH FIRST
18	SEARCH NEXT
19	DELETE FILE
20	READ RECORD
21	WRITE RECORD
22	CREATE FILE
23	RENAME FILE
24	READ LOG-IN STATUS
25	READ CURRENT DRIVE
26	SET BUFFER ADDRESS
27	READ ALLOCATION VECTOR
28	SET READ/WRITE STATUS
29	READ READ/WRITE STATUS
30	SET LIST ECHO STATUS
31	READ LIST ECHO STATUS
32	READ CLOCK (LOW)
33	READ CLOCK (HIGH)
34	READ DCB ADDRESS
35	TRANSLATE LOGICAL SECTOR TO PHYSICAL

4. Within a block, data is organized as records each of which is 128 bytes long. These records are logically contiguous but are physically located at a spacing on the disk which minimizes latency delays in transferring successive records within a block.

5. Access to data is controlled by a logical record number. The maximum number of records in an extent is 128 and thus the logically contiguous records in an extent are numbered from 0 ($00) to 63 ($3F) or 127 ($7F).

2.2.2.2 FCB DEFINITION

To keep track of the data needed for DOS/65 file operations, PEM uses something called a FILE CONTROL BLOCK (FCB). An FCB consists of 33 contiguous bytes numbered as bytes 0 through 32. The meaning of each byte is summarized in the following table:

Byte	Contents
0	drive number + 1 (i.e. 1 to 8) or 0 if DEFAULT
1-8	file name in uppercase, left justified, blank filled
9-11	file type in uppercase, left justified, blank filled
12	extent in binary
13-14	not used but reserved for future versions
15	number of records in extent (0 to 128)
16-31	block numbers (0 if non assigned)
32	next record to read or write (0 to 63 or 127)

In general the user must fill in bytes 0 through 12 before communicating with PEM so that PEM knows what file is being used. For example, if one wished to read a file named BASIC.COM from the DEFAULT drive, an FCB would be set up which would look like:

```
$00 $42 $41 $53 $49 $43 $20 $20 $20 $43 $4F $4D $00 $00 $00 $00
$00 $00 $00 $00 $00 $00 $00 $00 $00 $00 $00 $00 $00 $00 $00 $00
$00
```

If that FCB began at $1450, the user would OPEN the file by executing the following code:

```
     FCB  =    $1450
          LDA  #<FCB     get address with low in A and high in Y
          LDY  #>FCB
          LDX  #15       OPEN command
          JSR  PEM       execute it
```

If PEM found the file then the FCB after return from PEM might look like the following:

```
  $00 $42 $41 $53 $49 $43 $20 $20 $20 $43 $4F $4D $00 $00 $00 $43
  $1A $15 $16 $23 $24 $25 $26 $27 $28 $00 $00 $00 $00 $00 $00 $00
  $00
```

This indicates that there are $43 (67 in decimal) records in the file and that it consists of the 9 numbered blocks shown. If one then wanted to read the first record, the following code could then be used:

```
     LDA #<FCB      address of OPEN FCB
     LDY #>FCB
     LDX #20        read command
     JSR PEM        execute
```

After completion of that command, the last byte in the FCB would have automatically been incremented by one. Although perhaps not obvious, random access within an extent of a file is handled merely by setting byte 32 (the last byte) in the FCB to the desired record number before doing the read or write. For random access beyond an extent, the user must CLOSE the current extent (if open) and then must also calculate the correct extent and OPEN it if it is not already OPEN. Reads and writes of sequential records need not be concerned with the extent as DOS/65 will automatically OPEN the new extent if necessary.

One key question is where one goes to execute a command in PEM. Since PEM itself can be located at many different addresses in a system depending on the MEMORY SIZE, a standard location ($103) has been defined for a JMP PEM. That location as well as the complete memory organization of DOS/65 is discussed in the SYSTEM INTERFACE GUIDE.

2.2.3 SIM

SIM is the user peculiar interface between PEM and the users system. It consists of a set of JMP's which point to the routines which do the work and a block of data that defines the characteristics of the users console. The JMP's and the associated functions are listed in Table 2-2.

The precise action to be taken by each entry point and the contents of the CONSOLE DEFINITION BLOCK are defined in the SYSTEM INTERFACE GUIDE. It is important to note that SIM is the only portion of DOS/65 which must be modified by the user.

TABLE 2-2. SIM INTERFACE

ADDRESS	NAME	FUNCTION
SIM	CBOOT	Entry from BOOT to initialize system and execute CCM
SIM+3	WBOOT	Execute warm boot by reading CCM and PEM into memory from disk and then executing CCM
SIM+6	CONST	Check console status
SIM+9	CONIN	Read single ASCII character from console
SIM+12	CONOT	Write single ASCII character to console
SIM+15	LIST	Write single ASCII character to printer
SIM+18	PUNCH	Write single byte to serial device
SIM+21	READER	Read single byte from serial device
SIM+24	HOME	Move current drive to track 0
SIM+27	SELDSK	Set current drive to value passed
SIM+30	SETTRK	Set current track on current drive to value passed
SIM+33	SETSEC	Set current sector on current drive to value passed
SIM+36	SETDMA	Set disk buffer start address to value passed
SIM+39	READ	Read a sector from current drive, track, and sector into buffer
SIM+42	WRITE	Write a sector from buffer onto current drive, track, and sector
SIM+45	LISTST	Check printer status
SIM+48	CLOCK	Read real-time clock
SIM+51	XLATE	Translate logical to physical sector for current drive
SIM+54	CONDEF	Block of data defining console characteristics (Not a JMP)

APPENDIX A - SUPPLIED TRANSIENTS

The following transient programs are supplied along with the operating system itself.

LICENSED (COPYRIGHTED) PROGRAMS

EDIT.COM	Text editor.
ASM.COM	Two pass 6502 assembler.
MAKECOM.COM	Creates an executable file of type .COM from a code file of type .KIM as produced by the program ASM.COM.
SYSGEN.COM	Creates a "new" system incorporating the users SIM and BOOT.
UPGRADE.COM	Creates a Version 2.0 system when executed on a Version 1.2 system. (Special Order Only)
DEBUG.COM	Debugger.
COMPILE.COM RUN.COM	Compiler and interpreter for BASIC-E/65.

PUBLIC DOMAIN PROGRAMS

FORMAT.COM & .ASM	Formats disk for the UFDC-1 or OSI controller.
MOVE.COM & .ASM	Transfers a file from one location to another.
COPY.COM & .ASM	Copies all, data, or system portions of a diskette.
ALLOC.COM & .ASM	Shows disk space usage in visual and numeric form.
DISKTEST.COM & .ASM	Conducts non-destructive test of a diskette.
*xSnns.ASM	SIM source (x, nn, and s are configuration peculiar.)
*xBnn.ASM	BOOT source.
*xL.ASM	LOADER source.
BASICIO.ASM	I/O package source for OSI/Microsoft 6-digit BASIC.

TINYIO.ASM	I/O package source for Pittman Tiny BASIC (TEA =$200).
COMPARE.COM & .ASM	Compare two files on a byte-by-byte basis.
MORE.COM & .ASM	Transfer file to console in screen size pages.
MODEM.ASM D65TER.ASM	Telecommunications programs.

OPERATIONAL PROCEDURES

See separate manuals for EDIT, ASM, DEBUG, SYSGEN/UPGRADE, COMPILE and RUN.

MAKECOM

MAKECOM is executed by entering a command line of the form

```
(drive:)MAKECOM (drive:)(ufn)
```

where the (drive:) terms are optional drive specifications. The first is used to determine the drive (other than the default) from which MAKECOM.COM is to be read. The second determines the drive (other than the default) from which the source code file of type KIM is to be loaded and to which the executable file of type COM is to be written. If the type field of the ufn is blank, it is assumed to be KIM. In any event the actual type of the source code file must be KIM. The following are examples of valid LOAD commands:

```
makecom b:test.kim

a:makecom terminal
```

Since MAKECOM produces a contiguous machine code file which can be loaded at the TEA start as a transient, it will zero fill all areas not specified by the KIM file until the last KIM record is read. The KIM file must begin at an address greater than or equal to the TEA start address and must be monotonically increasing. If the user wishes to produce machine code files which begin and load at some address other than the TEA start, DEBUG must be used to load the file into the TEA with an appropriate offset. SAVE can then used to save the resulting machine code file. Such a file will not work correctly as a transient.

FORMAT (UFDC-1 and OSI Controllers Only)

Enter command of the form (drive:)FORMAT. The program will be loaded from the default or specified drive and action to be taken after that point will be prompted by the program. The diskette that will be formatted must be inserted into drive A unless otherwise prompted by FORMAT.

CAUTION

This program destroys diskette contents --- use with care!

MOVE

Enter command of the form:

```
(drive:)MOVE (drive:)ufn (drive:)ufn
```

The program will be loaded from the default or specified drive and will copy the file matching the second ufn to the file matching the first ufn. The optional (drive:) terms are used to specify drives other than the default. Unless the destination ufn (including drive) is the same as the source then the source file is not affected in any way by this command. If the first ufn consists of a drive specification only as in

```
move b: test.com
```

then the destination file will be given the same name as the source file. The following are examples of valid MOVE commands.

```
move data data (this moves file to itself)

move *testprog.old *testprog.com
```

COPY

Enter command of the form (drive:)COPY x where x is SYSTEM, DATA, or ALL. Program will be loaded from the default drive or the specified drive and the action required will be prompted by the program. COPY transfers an entire track at a time and verifies the copy. COPY can be used in a single drive system if the features included in the standard SIM are retained. In such a case diskette switching will be required for each track.

CAUTION

This program will overwrite all or part of the diskette in drive B - use with care.

DISKTEST

Enter a command of the form:

```
(drive:)DISKTEST
```

The program will be loaded from the default drive or the specified drive. All further action will be prompted by the program.

ALLOC

Enter a command of the form:

```
(drive:)ALLOC x
```

where x is a through h (without a trailing colon) and designates the drive to be reported. The program will be loaded from the default drive or the specified drive and will run to completion. ALLOC shows if blocks on the specified drive are available (0) or allocated (1) and shows the total number of unallocated blocks.

COMPARE

Enter a command of the form:

```
(drive:)COMPARE (drive:)ufn (drive:)ufn
```

The program will do a byte by byte comparison of the two files. If the files are identical the file length will be reported. If the files are unequal the byte number at which the files first differ will be reported. If the second (drive:)ufn combination consists only of the (drive:) specification then the first file name will be used. Examples of valid COMPARE usage are:

```
compare a:test.kim b:

compare filestat.bas filestat.bak
```

MORE

Enter a command of the form:

```
(drive:)MORE (drive:)ufn
```

The program will work like TYPE except that a screen of text will be sent to the console and then execution will stop until the user enters a SPACE or a CONTROL-C. A SPACE will display the next screen and a CONTROL-C will terminate the program and return to CCM. Examples of valid MORE usage are:

```
more sim.asm

more filestat.bas
```

MODEM & D65TER

These two programs provide the basis for sophisticated or simple telecommunications usage of a DOS/65 system. Both require user modification to handle the modem I/O characteristics.

APPENDIX B - ERROR MESSAGES

CCM ERROR MESSAGES

CCM detects and will report errors in one of several ways.

SYNTAX

If a command syntax error is detected, CCM will print a "?" on the console followed by the command line in which the error is detected. The kinds of error detected by CCM which fall in this category are:

1. AFN or UFN contains illegal characters

2. Transient COM file not found

3. AFN used where only a UFN is allowed

READ ERROR

If a file read error is detected during execution of a TYPE command, the message

```
"READ ERROR"
```

will be printed on the console. This most often means that a sector of the specified file is bad.

NO SPACE

If a SAVE command is executed and no room is available on the diskette a

```
"NO SPACE"
```

message will be printed on the console. The cure is to change diskettes or to erase unnecessary file from the current diskette.

CANNOT CLOSE

The message

```
"CANNOT CLOSE"
```

will only appear during execution of a SAVE command if an error is detected by PEM when an attempt is made to update the directory for the file. If this happens, check to make sure that the diskette was properly logged-in and that the allowable number of directory entries has not been exceeded due to the SAVE operation.

NOT FOUND

The message

```
"NOT FOUND"
```

will be printed if the DIR command finds no directory entries which match the specified AFN or UFN or if the file specified in a REN or TYPE command can not be found.

FILE EXISTS

If the new name specified in a REN command corresponds to the name of an existing file the message

```
FILE EXISTS
```

will be printed.

LOAD ERROR

This error will only occur during execution of a transient program and will result in the message

```
LOAD ERROR.
```

The most likely meaning is that the transient load activity tried to overwrite CCM with the program. If this occurs it means that the program is too large for the memory available.

CANNOT OPEN

If the message

```
CANNOT OPEN
```

is printed during execution of a SAVE command, it means that a disk error was detected during the file opening step. This is a highly unlikely error and if it occurs it indicates that the directory of the diskette may be in error.

WRITE ERROR

The message

```
WRITE ERROR
```

means that a disk write error has been detected during execution of a SAVE command. The most likely cause is exhaustion of the available space on the diskette.

PEM ERROR MESSAGES

PEM has only three error messages which it prints. They are as follows where x is replaced with the drive letter:

```
PEM ERROR ON x - BAD SECTOR
RET) TO IGNORE - (OTHER) TO ABORT

PEM ERROR ON x - R/O

PEM ERROR ON x - INVALID DRIVE
```

BAD SECTOR

The bad sector error will occur if SIM is unable to successfully read the specified sector. The user is given the option of ignoring the error or of aborting execution by performing a WARM BOOT. This is usually a serious error indication and is most likely due to a defect on the disk or a hardware failure.

R/O (READ/ONLY)

If a write is attempted to a drive marked as READ ONLY (R/O) by the system, this error will occur. The most likely cause of this error is an attempt to write to a diskette which was not the diskette in the drive when the last WARM or COLD BOOT was performed or when the last INITIALIZE SYSTEM (X=13) PEM command was executed. This error always causes a WARM BOOT to be executed.

INVALID DRIVE

An attempt to reference a drive outside the range of 0 to 7 (A to H) or a return value of 0 from the SELDSK routine at SIM+27 will generate this error message. This error always causes a WARM BOOT to be executed.

APPENDIX C - COMMAND LINE PARSING

When a transient is executed, CCM continues to scan the command line after the transient UFN. If possible, CCM will build up to two FCB's using the data. CCM will also transfer the full line after the UFN to the default buffer.

FCB BUILD

If CCM can build a valid FCB from the first contiguous character string after the transient UFN then the drive, name and type portions of the resulting FCB will be placed into the appropriate bytes of the DEFAULT FCB at $107. If the second contiguous character string is also a valid UFN or AFN the drive, name and type fields of the resulting FCB will be placed into the appropriate bytes of a FCB at DEFAULT FCB + 16 or $117. If this second FCB is to be used and if the FCB at $107 is to be used, then the FCB at $117 must be moved to another location or it will be destroyed by file operations using the FCB at $107. The following examples show command lines and the number of FCB's that CCM would build:

Command Line	Number of FCB's
`edit file.asm a:`	2
`proc`	0
`compile source`	1

NOTE

If a FCB can not be constructed based upon the command line contents, the appropriate portion of the DEFAULT FCB is filled with ASCII blanks ($20).

LINE BUFFER BUILD

Regardless of whether or not CCM can build any FCB's from the command line, it will place all of the command line after the transient UFN into a special buffer at DEFAULT BUFFER ($128). The buffer is similar to that used for the READ BUFFER command in PEM except that the first byte in the buffer is the number of characters in the buffer rather than the maximum buffer length. The text characters begin in the second byte. The following examples illustrate how this feature works:

Command Line	Buffer Contents
`proc =`	$02 $20 $3D
`compile source #`	$08 $53 $4F $55 $52 $43 $45 $20 $23

APPENDIX D - SINGLE DRIVE SYSTEMS

Several standard versions of SIM have included the capability to handle multiple "logical" drives even if the user has only a single physical drive, some users were not aware of that capability. If the user answers "1" to the opening question asking for the number of drives, then SIM will handle all subsequent references to "logical" drives A through H with a "MOUNT x" message if the next "logical" drive is different than the current "logical" drive. Once the user has placed the correct "logical" diskette in the drive in response to the "MOUNT x" message, a key should be pressed on the console to tell SIM that the appropriate diskette has been mounted.

Obviously some routines are not optimized for a single drive system. COPY is the most obvious example. Since COPY reads and writes only one track at a time, it will require many diskette switches to copy an entire diskette. The user who is limited to a single drive should consider modification of COPY to read and write as many tracks as will fit into the available memory.

Current SIM versions (3.xx & later) do not include the MOUNT capability. Users needing an earlier version should contact Richard A. Leary for a copy.

APPENDIX E - AUTOMATIC COMMAND EXECUTION

There are circumstances under which the DOS/65 user may want to automatically execute a CCM command at cold boot without any user input required. Such an approach is especially attractive for an end user system. The following procedure can be used to create a system that will automatically execute any valid CCM command line after a cold or warm boot is performed. This data is valid only for SYSGEN 2.15. Other versions will require a different address depending on code changes.

Step 1 Enter the following CCM command:

```
DEBUG SYSGEN.COM
```

Step 2 Using the S command insert the desired CCM command line in upper case ASCII beginning at TEA+$103A. Insert a $00 after the last ASCII character in the line.

Step 3 Using the S command insert at TEA+$1039 the number (in hexadecimal) of ASCII characters that were entered in Step 2. Do not count the $00 as one of the characters entered.

Step 4 Enter `ctl-c`.

Step 5 Enter the following CCM command:

```
SAVE 30 AUTORUN.COM
```

NOTE

The name AUTORUN is arbitrary. You may use any name desired but do not use any standard name (e.g., SYSGEN).

Step 6 Use AUTORUN instead of SYSGEN to create the correct size system, merge your BOOT and SIM, and then write the system to the desired drive. See SYSGEN manual for operational procedure but substitute the name used in Step 5 everywhere SYSGEN is used.

To use the new system a cold boot should be performed using the new system. A warm boot will also work as long as the system size is unchanged and SIM is unchanged.