STARDOS

STARPOINT SOFTWARE

Designed, Implimented and Produced by:

BRYCE NESBITT
BRUCE O. HAMMOND
SCOTT M. BLUM

STARDOS

STARPOINT SOFTWARE
6013 Macks Gulch Road
Gazelle, California 96034-9412
(916) 435-2371

Stardos Table of Contents

Introduction

Welcome to the wonderful world of Stardos. You are reading "Stardocs", the operating manual to Stardos, one of the most important additions that you can make to your Commodore Computer. Whether you use your computer for business, education, home finance, or just playing games, Stardos will help you get more out of your machine.

What is Stardos?

Stardos is an enhancement kit for Commodore 64 and compatible computers. Stardos is designed to make the user-interface (the way you talk to your computer) much easier to use, speed-up disk access, and provide instant access to commonly used programs and utilities.

Why was Stardos created? (what will it do?)

The Commodore 64 is probably the best value on the market for computing power per dollar. It has 64K of RAM, a powerful video chip, sound/music synthesizer and a full screen editor, at a price that can't be beat. All in all this adds up to a computer enthusiasts dream come true!

Unfortunately the C64 does have a few shortcomings, most serious, the disk drive interface is far too slow to be suitable for some applications. It also is tedious and verbose to send commands to the disk drive, read the directory or even to load a program. The full screen editor, however wonderful, is missing some commands that would make life simpler for those of us that use it.

So, Stardos was born. Stardos was conceived as a system to alleviate some of the aforementioned disk I/O lethargic tendencies and cure as many of the oversights, omissions, and bugs as possible. It also adds a plethora of new features, enhancements and utilities to make working and playing with the C64 even more productive and enjoyable than ever before.

What Stardos will not do

Stardos is not capable of fixing defective equipment. If your computer or disk drive system does not work, Stardos should not be attached until the problem is fixed.

Stardos is also not able to make bad software better, but it will probably speed it up considerably if it does a lot of disk access.

Lastly, if you are loading a commercial piece of software that has its own fast loading system, Stardos will enhance the loading speed of the software only to the point that the packages' own loaders have completely taken control of the loading process. Stardos will not disconnect, but will sit quietly until the

program once again attempts to access the disk drive in the normal, slow fashion.

The DOS wedge

The DOS wedge is just about the most useful added feature of your Stardos system. It is in fact SO useful that even if you read no further in the manual, and learn no other commands you should learn these.

History

The Commodore 64 is a direct relative of the older Commodore PET computer. The PET featured what is known as device oriented i/o; no special consideration is given to specific devices such a disk drives. You experience this on the Commodore 64 as a need to type ",8" after each LOAD command and the need to type "OPEN1,8,15," before sending a command to the drive. A program known as the "WEDGE" or "DOS WEDGE", written by Bob Fairbairn for the PET fixed all this. The wedge provided a quick and easy method for interacting with a disk drive. This useful utility is supplied on the TEST/DEMO disk shipped with Commodore disk drives. Unfortunately, it must be loaded each time you wish to use your machine, and, because of the way it is written, slows down any running BASIC program.

The Stardos Built in Wedge

StarDOS has a built in wedge that follows the original syntax, never needs to be loaded, and will not slow the execution of BASIC programs. At this point please put down the manual, reach over and turn your system OFF and then ON. When the power-up screen appears type:

```
@<RETURN>
73, CBM DOS V2.6 1541,00,00
```

The disk drive is telling you what version and type it is. If all goes well, try typing this:

```
@<RETURN>
00 ,OK,00,00
```

This is the normal state of things... this disk drive responds by telling you all about its problems, in this case it is 'OK'. Neat huh? Whenever the disk drive error light is flashing you can use this command to find out what the error is.

Sending Commands

You may also send commands with the wedge. Insert a disk, and type @I0. The disk will spin for a moment as the drive reads this disk. Here is a short list of legal commands (for a full description please refer to the manual that came with your disk drive).

```
@S0:filename[,filename,filename...]     ;scratch file(s)
@I0                                     ;initialize disk
@V0                                     ;validate disk
@C0:newfile=oldfile                     ;make copy of file on same disk
@R0:newname=oldname                     ;rename file
@N0:diskname[,id]                       ;format (erase) entire disk
@XL:filename[,filename,filename...]     ;lock file(s)
@XU:filename[,filename,filename...]     ;unlock file(s)
@XD:device                              ;renumber drive
```

The above drive commands may be sent from many commercial programs and from all of the Stardos function programs.

New and easy ways to access files and directories

Type @$<RETURN> (the $ means directory). The directory of the disk will scroll onto the screen, without disturbing the program in memory. Hit <RUN/STOP> to stop the listing or hold down <LEFT-SHIFT> to pause. After a directory is on the screen you can have even more fun. Cursor up to a line with a filename on it, type /<RETURN>. The selected file will load into memory. ^<RETURN> at the beginning of a line with a filename on it, will load and run the file.

Stardos contains shortcuts for the most common disk commands; note that quotes are optional around the filenames. The commands are as follows:

```
/filename       ;LOAD"filename"  (BASIC PRG)
↑filename       ;LOAD"filename" then "RUN"  (BASIC PRG)
%filename       ;LOAD"filename",dev,1 (M.L. or BASIC PRG)
<filename       ;SAVE"filename"  (BASIC PRG)
=filename       ;VERIFY"filename" (BASIC PRG)
!filename       ;type the contents of the file  "filename" to  the screen without
actually loading it  (any type except REL)
```

Other handy additional functions

£ (British pound symbol)

This will take you into the function menu of Stardos. From this menu, you may invoke such things as TEXTED (the text editor) or Starmon (ML/Drive monitor)

<C=><RUN>

This key combination provides quick and easy way to automatically load and run the first program on a disk.

@$device,device

Set default source and destination drives used in loads, saves, filecopy and diskcopy. If the second parameter is not given, it will default to 8 (normal first drive number.)

LOAD and SAVE commands may now be issued without a device number. This will be appreciated by those who are sick of appending ",8" to everything they do.

Example: LOAD"FISH" <RETURN> or /FISH <RETURN>

will load "FISH" from current device (drive 8 normally).

@#Q

Quit DOS wedge. This will disable the DOS wedge until a reset of the computer occurs.

Screen editor additions

The Commodore-64 comes stock with one of the best screen editors of any personal computer. Stardos provides some enhancements to make it even better.

<SHIFT/LOCK> or <LEFT-SHIFT> Pause screen scrolling.

<SHIFT><CTRL><CLR/HOME> This three-key combination works just like <SHIFT>-<CLR/HOME> except it clears from the cursor position downwards instead of clearing the entire screen.

<SHIFT><CTRL><INST/DEL> Deletes from the cursor position to the end of the line.

<SHIFT><CTRL><CRSR> Just like <CLR/HOME> except it places the cursor at the bottom left instead of the top left.

<CTRL></> Produces a CHR$(127). Some other computers that you might connect to via modem require CHR$(127) in place of the CHR$(8) that the normal delete key returns.

The Function Menu

The function menu is the central hub of all of the built-in Stardos programs and utilities. You may invoke the function menu by typing "_" from BASIC. From this menu you may select any of the Stardos function options as listed below.

Function Option 1, TEXTED the text editor

TEXTED is the built in editor for Stardos. It is intended for quickly creating small notes or memos and simple tasks like the creation of text files for modem uploading or short letters. If you are intending on editing anything of any size or if you need fancy printer control you need a real word processor like Kwik-Write(tm) or Paperback Writer(tm).

TEXTED is entered by typing < ><RETURN> (to get to the menu) followed by <1>. Please do this now. You should be looking at a screen with a status line at the top, and dots at the bottom which indicate the end of text. Since you have not entered any text, there are a lot of dots. As an introduction to TEXTED please type the following text;

"The infrastructure of JELL-O contains the secrets of the universe."

Now, General Foods corporation would like you to know that "JELL-O" is not a generic term, but rather a trademark. Move the cursor over the "c" in the word "contains". Now type "brand gelatin dessert ". You may have noticed that while you typed everything else moved out of the way. This is called insert mode. TEXTED automatically puts you in insert mode whenever you enter it. Now suppose that you don't like artificial coloring and can't imagine that anything containing it could also have universal secrets. With a text editor quotations are easy to change. Cursor up to the "J" in JELL-O. Now hit <SHIFT><INS/DEL>; this puts you into typeover mode. Now type "KNOX unflavor". Hit <SHIFT><INS/DEL> again and finish things off by typing "ed". That is all there is to it!
TEXTED has some other features, as outlined below:

<F1> Load a file

This loads a file from the current drive into memory (for more information how to set the current drive, refer to the DOS wedge section). The command asks for a filename and if successful loading it places it at the end of any text already in memory. If you don't want this to happen erase any text in memory first by exiting and reentering TEXTED.

<F2> Save a file

This saves the current file to disk. The command asks for a filename, which must be unique. If you wish to write over an existing file you may use the SAVE-@ feature by preceding the name with "@:".

<F3> DOS Wedge

This function allows access to the standard DOS wedge. From here you can send commands, list directory, read the error channel and change the current device number. See the DOS wedge section on sending commands for more information.

<F4> Type a file

This function allows you to "peek" at a file without actually loading it. The contents of the file will scroll by on the screen. To slow it down press <CTRL>, to pause press <SHIFT/LOCK>, and to stop press <STOP>.

ZoomUp/ZoomDown

The cursor keys are fine for moving around in small files, but when you really want to move a long way, use <CLR/HOME> key. Unshifted, it will move you down 10 lines, shifted it moves up 10 lines. If you forget which is which simply remember that this key works just like the up/down cursor key.

<F5> Print file

This option allows you to print the file currently in memory.

<F6> Enter Value

This function allows you to enter any number into your file and is usually used to imbed special printer codes that cannot be typed at the keyboard. Upon execution TEXTED will prompt you for a value; hit <RETURN> to forget the whole thing or the number you wish to add. Decimal is the default. To enter a HEX value precede it with a "$". Example:

```
VALUE:$1B      ;Enter the "ESCAPE" character into the text.
```

Advanced features

If you hold down the Commodore-key (C=) while entering TEXTED it will not clear memory. This means that theoretically you could exit TEXTED and get back to it without losing anything.

Texted stores text starting at $D000 and extending to the end of memory. If you happen to nuke your text you can save it into a file using Starmon. Type ":134" <RETURN> then "S" filename "8 d000 ffff" <RETURN> to dump the contents of memory to disk. You may then reload the file with Texted.

Binary files

Texted can safely edit binary files and will not change or modify any of the control characters in the file. When you position the cursor over a character in the file its value will be displayed in hex in the upper right hand part of the screen.

Function Option 2, Filecopy

The file copier of Stardos is a fast, powerful and simple to use (point and shoot style) file duplication utility. It is capable of copying all types of files and also files larger than 64k. It will not copy files or information that does not appear in the directory, to do that, you need to use option 3, Diskcopy. To activate the filecopy module, press "2" from the Stardos function menu. You will be presented with the filecopy display screen, simply a solid line at the top of the screen. If you wish to use two drives, change source and destination with the "@#" command documented in the DOS wedge section. The following is a summary of available functions:

<R>	Read directory and display files (source)
<C>	Copy selected files (source-->dest)
<Q>	Quit - Return to Stardos menu
<@>	Send DOS command (dest)
<CRSR>	Move blip up and down.
<SPACE>	Select/De-Select a file to copy.

<R> Read Disk Directory

This is almost always the first function invoked when using the file copy system. Place a disk in the source drive, and press the "R" key. After a brief pause, you will be presented with a list of files contained on that disk. If the disk contains more than 21 files, then only the first 21 files will be displayed; use the <CRSR> key to display additional files. At the top of the screen, the name of the disk and blocks free will be displayed.

If you find that there are no files that you wish to copy from this disk, insert another disk and hit <R>.

<C> Copy Selected Files

Before you select this option, you should have selected the files you wish to copy by using the <CRSR> key and pressing <SPACE> to select the file(s) that you wish to copy; have a formatted destination disk, with enough free space, ready to accept them.

You will be prompted to insert the source disk that contains the files to be copied (it is probably still in the drive.) At this point, hit any key except <SHFT> or <STOP> to begin the copy process. After reading all files (or as many as possible) you will be prompted at the top of the screen to insert the destination disk. Insert a pre-formatted disk that you wish to copy the files

to and press a key to write them. If the file copier was not able to copy all files on the first pass, you will be asked to re-insert the source disk, and the process will continue until all files are copied.

NOTE: If a file on the destination disk is of the same name as a file to be copied form the source disk, that file will be erased and the new file saved in its place.

<@> DOS wedge

Hitting @ will allow you to send a DOS command to the disk drive. This may be used to NEW (format) a disk, list the directory or to rename files before copying.

Function Option 3, Diskcopy, diskette duplicator

This option will allow you to duplicate unprotected diskettes quickly, easily and reliably using one or two disk drives (see wedge). The diskcopy program is not designed to copy protected programs. If this is a need, you should use a dedicated disk nibble system such as Di-Sector(tm).

After entering this module, you will be prompted for the SOURCE disk. Insert the disk you wish to copy and press <RETURN>. After the read pass is complete, you will be prompted to insert the DESTINATION disk. At this point, you should insert the disk that you wish the copy to be placed upon and press <RETURN>. The destination disk will be formatted and verified, all information previously on the disk will be erased. If a flaw in the destination disk is apparent after formatting, the copy process will be aborted, otherwise, the data from the read pass will now be written to the disk. If yet more data remains to be duplicated, then the process will request the source and destination disks to be swapped until all data has been copied.

When using two drives, you will be prompted for both the source and destination disks before the copy process has started. You will not need to swap disks.

You should always write-protect the source disk before duplicating it to insure that, in the event of an error, your original data will be intact.

Function option 5, Starmon, ML/Drive monitor/Sector Editing

This module is a very advanced machine language monitor which will allow you to write, debug, and edit programs of your own creation, or those of others.

Because of the array of features and simple command syntax, it is well suited to super-hackers or beginners (H.I.T.).

Notation Conventions in Monitor Documentation

Numeric is any of the following:

Hexadecimal: consisting of an optional '$' followed by hexadecimal digits.
Examples: $2020, 3F, $1ABD

Decimal: '!' followed by decimal digits.
Examples: !21, !32767, !100

Octal: '&' followed by octal digits.
Examples: &37, &10

Binary: '%' followed by binary digits.
Examples: %11001010, %1100

Operators: Any numeric can include the math operators + and -. For instance $100-!25 is a valid numeric, with the value of 231 decimal.

<start-address> is any valid <numeric>, between $0000 and $FFFF.

<end-mark> is a numeric address indicating the end of a memory range. The syntax can be expressed with either an absolute address or a comma followed by a numeric length.
Example: M 8000 8100 or M 8000,100
will both display $100 bytes of memory beginning at location $8000.

<byte> is a numeric between 0 and $FF or text that is inside of quotes, which will generate one <byte> per ASCII text character.
Examples: $34, 2D, !100, or &37.
Example of text: "ABCD" which generates the bytes $41 $42 $43 $44.

<device> a valid numeric device number between 0 and 15 decimal usually 8 for the first disk drive.

Note that operandi in brackets [and] are optional, and will default to various values depending upon the command.

Starmon has 8 command classes: Memory, Assembler, Execution, Register, Redirection, File manipulation, Sector Editing and Miscellaneous.

Summary of Starmon Commands:

Memory Commands

: Set Memory
F Fill Memory
T Transfer Memory
I Interpret as ASCII
H Hunt memory
C Compare memory

Assembler Commands

A Assemble bytes
; Assemble a single line
D Disassemble bytes
U Undefined op-code disassembly

Execution Commands

G Go
J JSR

Register Commands

R Register Display
* Register Modify

Redirection Command

O Operate

File Commands

L Load
S Save

Sector Editing

↑R Read sector
↑W Write sector
↑N Next file sector

Misc Commands

X Exit
= Base conversion/Math
@ Dos Wedge

Commands that operate on memory

The following commands allow you to display, and modify memory:

: (set memory)
Fill memory
Transfer
Memory display
Interpret memory as ASCII
Hunt for pattern.
Compare memory.

SET MEMORY Syntax: : <start-address> <byte> [<byte> <byte>...]

The set memory command will begin modifying memory at <start-address> for as many bytes as are given on the command line. Examples:
':4000 10 &21 %10101 "HELLO"' Will set memory from $4000 with the hexadecimal values $10, $11, $15, $48, $45, $4C, $4C and $4F.

FILL MEMORY Syntax: F <start-address> <end-mark> <pattern>
where <pattern> is a string of up to 32 bytes. Examples:

'F C000 C100 "PATTERN " 10 %1010' Will fill the memory from C000 to C100 inclusive, with the repeating hexadecimal bytes $50 $41 $54 $54 $45 $52 $4E $20 $10 $0A.

'F !2048,8 &37' Will fill the memory from 32767 decimal, for 8 bytes with the value 37 octal.

TRANSFER MEMORY Syntax: T <start-address1> <end-mark> <start-address2>
Transfers bytes from <start-address1> through <end-mark> to the memory starting at <start-address2>. Examples:

'T 8000 8100 4000' Will move the bytes residing at $8000 through $8100 and duplicate them in the block $4000 through $4100.

'T 1000,!100 $9000' Will move the bytes residing at 1000 hex, for 100 decimal bytes, to location 9000 hexadecimal. Note that the '$' in the '$9000' is optional.

Note that this command is especially useful in conjunction with the 'O' command explained later, for copying RAM to and from the disk drive.

MEMORY DISPLAY Syntax: M [<start-address> [<end mark>]]
This command will display bytes from memory starting at <start-address> and continuing until <end-mark>. If no <end-mark> is given, it assumes 1 screenfull of data. If no <start-address> is given, it commences from the last address displayed. This command will always display at least eight bytes, and rounds the number of bytes displayed up to the next multiple of eight. Examples:

'M 8000' Will display one screenfull of data in hexadecimal, and ASCII starting at 8000 hex.

'M D020,!64' Will display 64 decimal bytes beginning at hex location D020, in hex and ASCII.

'M' This, if executed directly after the M D020,!64 command, will display the next screenfull of memory,
which will begin at D060.

INTERPRET ASCII Syntax: I [<start-address> [<end-mark>]]
This command will display the memory selected in ASCII. If no <end-mark> is specified, one screen full will be displayed. This command, similar to 'M' will display at least 32 bytes, and will round up to the nearest multiple of 32 bytes. Examples:

'I 8000' Will display from 8000 hex, one screenfull of memory in ASCII.

'I !4096,!96' Will display 96 decimal bytes of data starting at 4096 decimal, in ASCII format.

HUNT MEMORY Syntax: H <start-address> <end-mark> <pattern>
This command will search through memory searching for <pattern>, which may include question marks for any position, which is a wildcard match. Examples:

Assume that memory from $4000 contains the following data:

```
:4000 34 22 11 45 21 56 3F 1A
:4008 2B 33 4F C2 19 24 2E 5F
```

'H 4000,F 1?' Will return all of the addresses between 4000 and 400F where the top nibble equals 1, in this example, 4002, 4007, and 400C.

'H 4000 7000 "F?SH" %1???????' Will Find such things as "fish", "fosh", "fesh" etc. as long as they are followed by a byte with the hi bit set.

'H 4000 7000 %01??????' Will return all of the addresses in the range which contain bytes which have bit 7 reset, and bit 6 set.

This is probably the most useful location for the binary numerics, as it allows you to search for specific bit patterns or set of instructions.

COMPARE MEMORY Syntax: C <start-address> <end-mark> <compare-address>
This command will compare two areas of memory for differences. If a byte between the <start-address> and <end-mark> does not match a corresponding byte in the comparison area, then its address will be printed.

NOTE: This can be quite a list if you attempt to compare two unrelated areas of memory; press stop to abort.

Commands that manipulate Assembly Language

There are 4 commands that allow you to program in assembly language.

A Assemble
D Disassemble
U Undefined op-code disassemble
; Assemble a single line of code

ASSEMBLE Syntax: A <address> <op-code> [<operand>]
This will begin assembling code at the address you specify. Examples:

```
A 8000 LDA #$41
```

Will begin assembly mode, inserting the instruction LDA #$41 at location 8000. The monitor will then print on the following line, the next address, whereupon you have the option of hitting return again, and exiting assembly mode, or typing in another op-code such as:

```
A 8002 LDX #$04
Continuing...
A 8004 SEC
A 8005 DEX
A 8006 LDA #"A"
```

A 8008 <ret>
.

The underlined text indicates computer output. If an illegal op-code, or the wrong operand for an op-code is attempted, the monitor will print a question mark at the point where the syntax error occurred, and assembly will be aborted. Undefined op-codes may be used in the assembler. This interactivity is well suited to the beginning programming student. Also note that in the operand field, the default base for numerics is decimal (!), and not hex ($).

For more information on assembly language, consult a book specifically written for that purpose.

DISASSEMBLE Syntax: D [<start-address> [<end-mark>]]
This will begin disassembling code in memory, displaying it in hex, 6502 instructions, and at the right-hand edge of the screen, ASCII. Illegal instructions are marked with ??? in the op-code field. Note that at the beginning of each line is the character ;, which will allow you to modify your code with the screen editor, and automatically re-assemble it. If an instruction passes over the end mark boundary, it will complete the display of that instruction. If the end-mark is omitted, the disassembly will continue

for 1 screen. If the start-address is omitted, the disassembly will commence from the current address.

UNDEFINED OP-CODE DISASSEMBLY Syntax: U [<start-address> [<end mark>]]
Identical to Disassemble, excepting that it will disassemble certain undefined op-codes. See Appendix for a list of valid undefined op-codes.

ASSEMBLE A SINGLE LINE Syntax: ; op-code [operand]
This command is for convenience of modification with the full screen editor. When doing a disassembly a ";" is printed at the beginning of each line to facilitate modification of M.L. statements. To change an instruction, simply type over it and press return.
WARNING! typing an instruction longer in bytes than the current instruction will cause the next memory locations to be destroyed.
Commands which execute code
There are two commands that allow you to execute portions of your code. They are:

Go, and JSR (Jump SubRoutine)

GO Syntax: G [<address>]
Where [address] is any valid numeric indicating program execution entry point. When this command is executed, the program counter is loaded with <address>, if given, and execution is passed to that location. If no address is given, execution will continue with the current value of the program counter. Program execution will continue until it locates a BRK instruction ($00) or until the operator taps the <RESTORE> key. If the program encounters the BRK instruction, the monitor will print out the registers, and the program counter will point to the BRK instruction. If the <RESTORE> key was pressed, the monitor will return with the program counter pointing to the instruction that was to be executed before it was interrupted. Examples:

'G !32768' Will begin execution at 32768 decimal.

'G' Will begin execution at current value of the PC (program counter.)

JSR Syntax: J [<address>]
This command is identical to GO, except that control will be returned to the monitor at the next RTS instruction. This is useful for testing subroutines.

Commands that involve the registers

There are two commands that allow the manipulation of registers. Register display, and * Set Registers

REGISTER DISPLAY Syntax: R

This command will print out the contents of the program counter, the accumulator, the X register, the Y register, stack pointer and the flags. Example:

```
R

   PC  AC XR YR SP  NV-BDIZC
*2020 04 34 21 A9 %01000010
```

SET REGISTERS Syntax: * [parameters]

This command is designed to be used with the register display command. Note that when the registers are displayed, the first character on the line is a '*'. This facilitates using the screen editor to modify registers. Example:

'R'

```
   PC  AC XR YR SP  NV-BDIZC
*8020 10 23 34 B3 %01001001
```

The registers and flags can be changed by typing the new contents in the respective fields, and pressing the return key.

Memory redirection

OPERATE Syntax: {see below}

The 'O' command, which stands for Operate, allows you to write code for your disk drive. The command takes one of three forms:

O <RET> which displays the current Input and Output memory settings.
O <i/o device> which sets both input and output to the same device memory.
O <input device> <output device> which allows a separate input device and output device. By example:

'O 8' Will set all input and output to go to device 8
Thus - 'M 1000' will display one screen of the drive memory.

'O 8 0' Will set input from drive and output to the computer.
Thus - 'T 1000,100 4000' moves $100 bytes from drive memory location $1000 to $4000 in the computer memory.)

This feature is for programmers wishing to write custom code for their disk drive, or other intelligent peripherals. All memory commands will work with re-direction, as will execution and assembly commands. Register commands, however will not work.

Commands for Loading and Saving Files

The two commands Load and Save are for loading machine language programs from disk, and saving them to disk. Load and save only operate with computer memory. Memory redirection has no effect.

LOAD Syntax: L "<filename>" <device> [<start-address>]
This command will go to the disk specified by <device> and attempt to load <filename>. If no <start-address> is given, then the <start-address> that the file was saved with is used. If a <start-address> is given, it will load the program at that location. Example:

```
L "FISH" 8
```

Will go to the disk, and attempt to locate the file FISH and load it into memory at its default location. If the file is not found it will leave memory unmodified and return with a prompt. If the file is found, it will be loaded into memory and the prompt returned.

L "FROG" 8 2020
Will go to the disk, attempt to locate the file FROG and load it into memory starting at 2020 hex.

SAVE Syntax: S "<file name>" <device> <start-address> <end-mark>
This command will save the bytes between <start-address> and <end-mark>, minus one byte, under <filename> on <device>. If the file already exists, it will not be over-written, an error will result. Example:

```
A 4000 INC $D020
A 4001 JMP $4000
A 4004 <ret>
.S "FISH" 8 4000 4005
SAVING "FISH"
.
```

Sector Editing

The monitor allows for a simple yet very powerful method of reading, editing and writing sectors directly to the disk. The sector editing commands are activated by typing ↑ (up arrow) the single command letter and it's parameters if any. In all cases, "drive" refers to the current drive (default 8); see the DOS-WEDGE section for additional information.

Note: Be sure to work only with backup copies of disks. Never work with one-of-a-kind originals or your only copy. Copy the disk with the disk duplicator first and work with the backup disk.

READ Syntax: ↑R [<track> <sector> [<addr>]]

This command will read a sector from the drive and place it in the computer's memory at <addr>, read and display the error channel and display the sector in ASCII and present you with the ↑R prompt. If <addr> is not specified then the current editing <addr> is used; the default is $CF00. If <track> and <sector> are omitted (i.e. hit <RET> on ↑R prompt) then the next physical sector will be read from the drive. After the sector is in memory, you may use any of the functions of the monitor to display and modify the data.

WRITE Syntax: ↑W [<track> <sector> [<addr>]]

This command will write a sector currently in memory to disk. If <addr> is omitted, then the current editing address is used (default $CF00). If <track> and <sector> are omitted then the data will be written to the last track/sector read.

Warning! Be careful when writing the data to disk and omitting parameters. Be certain that you have not changed the default settings since reading the data from disk.

NEXT Syntax: ↑N
This command will read the next sector of a file. The monitor obtains the next sector by reading the first two bytes of the current <addr> buffer and using them for <track> and <sector> respectively.

For a complete discussion of disk layout and theory, refer to the user manual for your disk drive.

Miscellaneous Commands
There are, finally, four miscellaneous functions, @ which allows you to execute DOS commands, = which does base conversion and math, memory banking and X for exit.

DOS WEDGE Syntax: @ <dos-command>
This command allows the user to execute one of several DOS commands. For a complete list of DOS commands, and how to use them, refer to the DOS wedge section of this manual. Example:

'@$' Will return the directory information for the current drive.

'@N0: FISH,SA' Will format the selected disk, naming it FISH and giving it the ID code, SA.

BASE CONVERSION/MATH Syntax: = <numeric>
This routine will convert one word (two bytes) from any of the valid bases to all of the other bases. Example:

```
= !32767
  $7FFF !32767 %01111111 11111111

= 3+"S"
V  $0056 !86 %00000000 01010110

= &40
```

$0020 !32 %00000000 00100000

Memory Bank Switching

By changing location $0001 of the computer's memory, you may control the ROM/RAM configuration. The default value is #$37; it is Kernal in, BASIC in and I/O in. Another popular configuration is #$34; Kernal out, BASIC out, I/O out (64K RAM). Bank switching will affect and function with all functions of Starmon including redirection, LOAD/SAVE, sector editing etc.

EXIT Syntax: X[C]

This command will return you to BASIC via the BASIC "warm start". When using monitor in sector editor, you will return to the edit sector screen.

If "C" is appended to the "X" then the system will do a cold reset to BASIC.

Appendix: Trouble Marksmanship

Problem: The disk drive does not work, or works slowly when I have sprites on the screen.
Explanation/Solution: The Commodore 64 serial bus system does not work with sprites on the screen. The same limitations apply to a machine with Stardos. With sprites on the screen, the serial bus will crash on average, once every 50 accesses. In order to ensure that your program will operate properly you must turn sprites off with a POKE 53269,0 before every load, save, disk, printer or other serial bus access.

Problem: The commands added by some BASIC extension products will not work.
Explanation: The Commodore 64 computer was not designed to accept extensions to its BASIC language, and therefore no interface standards or conventions were ever established for this procedure. The Stardos DOS wedge places a hook into the BASIC tokenizing routine, other programs may neglect to chain this vector through their own code, or may get confused by it.
Solution: Type @#Q<return> to disable the built in DOS wedge before loading the other program.

Problem: The DOS wedge will not function after certain programs have been loaded.

Explanation: Certain programs that extend the BASIC commands set may disable and/or replace the built in DOS Wedge, this is normal, and can only be changed by modifying the program.

Problem: I/O ERROR #5 ;Same as "device not present". If you get this you probably specified the wrong device number, or forgot to turn on the device you where trying to talk to.
Problem: I/O ERROR #4 ;Same as "file not found".

Problem: One of my programs will not load.
Explanation/Solution: This situation is rarely attributed to Stardos; we have made every effort to ensure that commercial software will continue to load and operate normally, however, some obscure software protection schemes may cause a problem for Stardos. If you suspect that Stardos is interfering with the loading of a disk, a Stardos equipped 1541 drive may completely be disabled by the following command: "@XH". This will tell Stardos in the drive to "hide". The Stardos ROM will no longer be electrically attached to the drive and will not interfere. This will cure the problem 99.99% of the time; you will have slow disk access, but the wedge and screen editor enhancements will still be in effect. The need to 100% disable Stardos is extremely rare and in the development/testing period has never arisen. None the less, the procedure is simple and temporary: Turn the computer off and unplug the Stardos from the

cartridge expansion slot. Turn the computer back on and type 'OPEN1,8,15,"XH":CLOSE1'. Stardos is now gone and will not interfere. If you are still having a problem with the loading of a program, suspect a damaged diskette or a slightly misaligned disk drive.

Appendix II: 6502 Undefined opcodes

Inst--	Abs	Abs,X	Abs,Y	Zer	Zer,X	Zer,Y	(Ind,X)	(Ind),Y	Imm	
ASO (ASL,ORA)	0F	1F	1B	07	17		03	13		0B
RLA (ROL,AND)	2F	3F	3B	27	37		23	33		2B
LSE (LSR,EOR)	4F	5F	5B	47	57		43	53		
RRA (ROR,ADC)	6F	7F	7B	67	77		63	73		
AXS (STX,STA)	8F			87		97	83			
LAX (LDX,LDA)	AF		BF	A7	B7		A3	B3		
DCM (DEC,CMP)	CF	DF	DB	C7	D7		C3	D3		
INS (INC,SBC)	EF	FF	FB	E7	F7		E3	F3		
ALR (AND,LSR)										4B
ARR (AND,ROR)										6B
XAA (TXA,)										8B
OAL (TAX,LDA)										AB
SAX (DEX,CMP)										CB

NOP	1A,3A,5A,7A,DA,FA
SKB	80,82,C2,E2,04,14,34,44,54,64,74,D4,F4
SKW	0C,1C,3C,5C,7C,DC,FC

```
ASO: ASL then ORA the result with A.
RLA: ROL then AND the result with A.
LSE: LSR then EOR the result with A.
RRA: ROR then ADC the result with A.
ASX: Store result A. AND X.
LAX: LDA and LDX with same data
DCM: DEC memory and CMP with A.
INS: INC memory then SBC the result from A.
ALR: AND A. with data then LSR
ARR: AND A. with data then ROR
XAA: Store X. AND A. in memory
OAL: ORA with $EE, AND with X. Store result in X.
SAX: SBC data from A. then AND with X, Store result in X.
SKB: Skip byte (i.e. branch +1)
SKW: Skip word (i.e. branch +2)
NOP: Skip work (i.e. Do very little)
```

Appendix III: Stardos Transfer Routines/Developer Support

Stardos is unique in that instead of patching around the 1541's slow speed like most other "fast loaders" it addresses the true cause of the problem; the low level i/o routines that read and write a single byte to and from the serial bus were very poorly designed and implemented.

Stardos provides a new set of low level i/o routines that are capable of operating at ten to fifteen times the speed of the old, slow method yet work with the same hardware and are fully forward and backwards compatible with them. This means that a Stardos C64 will work with either slow or Stardos equipped disk drives. Likewise a Stardos disk drive will work with either a slow or Stardos equipped C64s. The process of deciding which is which is called auto-detect. It works through a series of ID pulses sent hidden in otherwise normal TALK/LISTEN calls. The rest of this section deals with the details of how this is done and how developers of serial bus peripherals can take advantage of this.

The attention line is used to tell all devices connected to the bus to listen; such bytes are always sent by the old, slow method. When a computer that has Stardos capabilities wishes to talk fast it adds special fast-ID just before the first bit is clocked out. This pulse indicates to any Stardos equipped devices listening on the serial bus that the computer has Stardos capabilites. If a Stardos equipped device is addressed it will check to see if

anybody else is listening and if not send a special fast-acknowledge pulse after the last bit is clocked out. Until the next UNTLK or UNLSN, transmission will clip along at the speedy Stardos rate.

The system works quite well, allowing users to switch between any number of slow and Stardos equipped drives at will. The transmission routines are fast and reliable; given the hardware limitations of the 1541 drive we doubt there is any way to significantly improve upon them. Stardos transfer is compatible with the new 1571 type FAST transmission, however, unlike the 1571 format, it does not require any additional hardware. The ultimate would be a drive that is compatible with all three formats; SLOW standard, Stardos speed and 1571 fast. If you or your company manufactures a device that attaches to the Commodore serial bus such as a disk drive, hard drive, or IEEE interface we want to help you add Stardos capability! With a little over 100 bytes of code and a modest amount of effort by developers users with a C64 or C128 can take advantage of high speed Stardos transfer with your device. Example source code, licensees and technical help will be provided free of charge if you qualify and ask nicely. Please contact:

Stardos Developer Support
Starpoint Laboratories
6013 Macks Gulch Road
Gazelle, Ca 96034-9412

Appendix IV: How to be compatible with Stardos

From a programer's point of view Stardos is easy to live with. All ROM hooks and vectors point to the same place and all the routines, while faster, perform the same way. There are a couple of common-sense things to watch out for:

1. Sprites and the serial bus don't mix. With sprites on the screen, an unmodified Commodore 64 will crash on the average of once for every 50 disk accesses. Play it safe and always turn OFF sprites before any disk or printer i/o.

2. Stay away from computer location $A3. This location is reserved for use by the serial bus routines on a normal Commodore 64. In addition to its normal function Stardos keeps some important flags here. You should also be careful around the other locations that the low-level I/O routines use. The routines are TALK, TKSA, ACPTR, UNTLK, LISTEN, SECOND, CIOUT, and UNLSN. The locations used are $90, $94, $95, $a3, $a4, $A5 and $02A1.

3. Drive code: Stay away from location $7B. This location contains important flags. You should also avoid the other serial bus locations from $79-$7D and $96-$98. Remember that Stardos is faster... sector reads, GCR conversion and

transfer will happen at a different rate. Don't get locked into a time-dependant loop. "Double header" protection schemes should use a random delay between sector reads.

4. Don't assume the ROM's are completely identical. In order to work Stardos patches into the normal ROM; all entry points are identical, but the interior of the routines may have changed. Do not load values from the ROM and compare them with constants.

5. Do not duplicate Kernal code in your own code. Certain programs like Seven Cities of Gold, Heart of Africa, Flight Simulator II and Mindshadow have a very strange way of doing disk access. The programmers actually copied Commodore's old, slow clunky transfer routines and placed them in their own code. In effect this creates a Custom Slow-Loader! No speed or efficiency increase is realized by this method, yet it forever stunts the speed potential of the program. Why this was done is something of a mystery. The programers could have saved a lot of work by simply banking the kernal in (with LDA #$37; STA $01) then performing the required disk i/o and banking the kernal back out (with LDA #$35; STA $01). This would be faster, take less code, less work and be more compatible.

Glossary of terms

address The location of particular part of memory.

ASCII Acronym for American National Standard Code for Information Interchange. ASCII maps letters (like "A") to numbers (like 65). The C64 uses PETSCII, a rather mangled flavor of ASCII with upper and lower case characters reversed.

binary Refers to the Base 2 numbering system.

block 256 (100 hex) bytes.

bug An error, glitch, defect, gross boo boo, or screw-up in a program or hardware (an undocumented feature).

byte A unit of storage. One byte can hold a single ASCII character or value, for example the letter "Z" or the value $fA.

character One alpha-numeric unit of information, the letter "X" and the number "9" for example. Usually takes up one byte.

chr$ Abbreviation for character string. Usually followed by a number in parentheses which refers to the corresponding ASCII character. Examples: chr$(1) = Ctrl-A, chr$(54) = "6", chr$(88) = "x".

control character A character between chr$(1) and chr$(26). Example: chr$(4) is Ctrl-D, chr$(24) is Ctrl-X.

copy protection A scheme deliberately used by software companies to annoy users and make their programs incompatible with Stardos. If you have reason to believe that they have succeeded, use the "@XH" command, which will tell Stardos to "hide". This should overcome any difficulty. (Unless you have difficulty being patient enough to wait for the program to load!)

cursor A marker on the display screen that indicates where the next character will be entered, replaced, or deleted. You may normally move the cursor with the cursor-keys (marked <CRSR> on your keyboard).

debug The process of exterminating bugs.

dev An abbreviation for "device number". Most of the time this will be either 8 or 9 and refer to a disk drive.

DOS Acronym for Disk Operating System, the machine language program that allow the computer and drive to function.
DOS wedge A utility program, built into Stardos, that allows easy interaction with the disk drive.

diskette AKA "disk". Refers to those funny square things with the hole in the middle.

drive Shorthand for "diskette drive". Unless you are exceptionally kinky, this is where you stick your diskettes when you wish to access them.

execute To perform a computer instruction or program. Not "put to death".

file A group of related bytes all stored together, usually on disk.

filename A string of characters necessary to identify a file.

function menu A list of all the built-in programs and utilities available with Stardos. Press " " to get to this menu.

hexadecimal Refers to the base 16 numbering system. Often abbreviated "hex" and/or denoted by preceding with a "$".

H.I.T. Acronym for "Hacker in training".

memory Temporary or permanent data storage area that the computer may access directly.

monitor A utility program that allows direct access to memory and programming of the microprocessor.

nuke To destroy, demolish, obliterate, wipe out, mung, hash into little bits, waste, screw up, or make FUBAR, by means of atomic weapons, or with a computer.

octal Refers to the base 8 numbering system.

opcode One or more bytes of information which make up a machine language instruction.

parameter A value or string which is used with a command.

RAM Where your computer stores programs and data temporarily. Acronym for Random Access Memory.

ROM Where programs such as Stardos live their entire lives. Acronym for Read Only Memory.

syntax The order and form in which commands and parameters are to be specified.

wedge A general term for a program that extends the BASIC command set by "wedging" into memory. In Stardos it generally refers to the DOS wedge utility.