CARE AND HANDLING

Keep the enclosed magnetic recording away from magnetic fields, transformers, power supplies, motors, etc. so that the program will not be erased. Always protect magnetic cassettes from temperature and humidity extremes.

COPYRIGHT

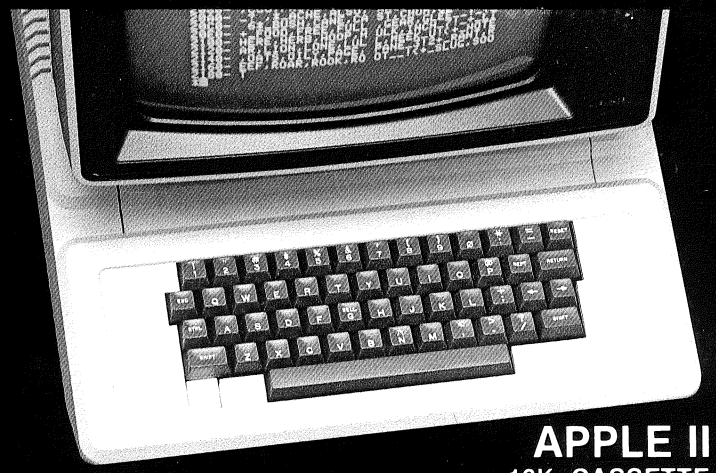
LIMITED WARRANTY

IMAGE Computer Products, Inc. shall have no liability or responsibility to purchaser or any other person or entity with respect to any liability, loss or damage caused or alleged to be caused directly or indirectly by this product, including but not limited to any interruption of service, loss of business and anticipatory profits or consequential damages resulting from the use or operation of this product. This product will be exchanged if defective in manufacture, labeling or packaging, but except for such replacement the sale or subsequent use of this program material is without warranty or liability.

IMAGE
COMPUTER PRODUCTS, INC

MADE IN U.S.A. FORM: ICP-2006 PRINTED IN U.S.A.

IMAGE
COMPUTER PRODUCTS



APPLE II
16K, CASSETTE

Monitor Extender™ 2006

This utility program works in complete harmony with the Apple monitor to extend your computer's capability and help you use the full power of machine language programming.

Screen display shows memory in HEX, ASCII or BINARY. Move data anywhere in memory without regard to direction or overlapping and read or write any sector on disk. Insertions may be in HEX or ASCII so you can easily format high speed text displays without conversions.

Study, modify or disassemble any program, complete with labels. Several programs may be combined, and the entire disassembled text file stored on disk/tape for later assembly. The slow listing feature steps through listings with ease.

THIS CASSETTE PROGRAM REQUIRES A CASSETTE PLAYER, 16K MEMORY.

Monitor Extender 2006

Programmed by Glenn R. Sogge
Program Edited by Al Baker
Directed by Dick Ainsworth

Monitor Extender
Table of Contents

Introduction

Monitor Extender is an extension to the normal commands of the Apple-II monitor ROM. You should be familiar with the normal modes and formats of the Apple-II monitor to most effectively use Monitor Extender. Normal monitor conventions are followed as much as possible to make the use of this code as simple as possible.

The Monitor Extender was written as a result of many hours of poking around in the Apple and various programs running on it. Being a "hacker" by nature, I like to figure out how things work. Since many programs come without much documentation, the only way to figure them out is to go looking at their guts. The various routines of the Monitor Extender are tools to aid the deciphering and modification of programs to make them more usable or flexible.

In all cases, the routines grew out of a particular problem on some project. Either the needed routines did not exist in the Apple monitor or they were not easily accessible from the keyboard command level. In some cases, several existing routines needed to be hooked together in new ways to achieve the desired result. After writing several small, special purpose programs, I quickly realized that an integrated collection of such utilities could save many hours of re-inventing - the - wheel even if only one or two of them were used at any given time.

Because programs and data areas can come in many places and sizes, a position independent version of such a utility pack seemed desirable; the Monitor Extender was always ending up in just the wrong place for a particular application. This was greatly expedited by the many extremely useful routines in the Apple monitor that could be called as needed. Without this resident monitor (particularly compared to machines like the TRS-80 that don't have a resident monitor), the writing of this program would have been extremely difficult and the result would have been much less flexible.

Using Monitor Extender

The best way to become familiar with the use of Monitor Extender is to try out the examples given in the following sections. In all the examples following, the Monitor prompt will be shown ("*"), but should not be typed in by the user. A carriage return will be indicated "(cr)". The token "(ctrl-Y)" means to press the keys marked "ctrl" and "Y" at the same time. Spaces should not be typed unless indicated by a "(sp)".

The code for Monitor Extender is position independent and will run from any 1.5K block of RAM. Details on how to move and run Monitor Extender from various locations are contained in the appendix.

When Monitor Extender has been started by typing in the Monitor command 800G and pressing RETURN, it does some initialization and then returns to the monitor with the "*" prompt and a beep. You can now begin to use the commands. None of the normal Apple-II commands have been changed, but some of them have been extended in function.

AUTHOR

Glenn R. Sogge is a 30 year old former composer with a degree in Art and 7½ years of retail business experience who has become fascinated and infatuated with those electronic crossword puzzles that are called computers. His most recent obsession is the implementation on the Apple of that extremely compact, fast, and user-extendable language / compiler / editor / assembler / operating system called FORTH.

He currently lives in Chicago, but pines for the Great Northwest where he grew up. To occupy and comfort himself until he can go west again, he studies computer architecture and language theory, the application of computers to real-time music performance systems, recent literature, and philosophy — especially the work of Wittgenstein and Frege.

How to load Apple single machine lanquage program cassettes.

1. If you have the autostart ROM, and an Apple Disk II, insert a disk in drive 1.
2. Turn on the computer and TV or Monitor
3. If you see an asterisk (*) cursor, go to step 3. If you have autostart ROM, press CONTROL C, press RETURN, type CALL-151, and press RETURN. You should see an asterisk (*) cursor on the screen.
4. Follow the instructions in your computer's instruction manual to connect your cassette recorder to the Apple II.
5. Insert the cassette tape into the recorder. Press the REWIND button on the cassette recorder and rewind the tape fully.
6. Remove cable from earphone jack on cassette recorder.
7. Press PLAY and listen for leader tone.
8. Stop recorder the instant you hear the tone and reinsert the cable in the earphone jack.
9. Type the instruction to load the tape. Do **not** press RETURN. The correct instruction is the top line printed on the right side of the cassette label (400.CFFR).
10. Press recorder PLAY button and then RETURN key. If the program has loaded properly, you will hear a beep and the "*" prompt and blinking cursor will appear approximately 30 seconds after the title screen shows.
11. Type the instruction to run the program and press RETURN. This instruction is the second line printed on the right side of the cassette label. The program will begin automatically. Run the program. Type 800G and press RETURN. The "*" prompt will reappear with a beep.

In case of difficulty:

Programs from cassettes do not always load correctly the first time. Static or even brief interruptions of power can cause errors. If your computer prints the word ERR on the screen, or you do not hear the beep after two or three minutes, try reloading the tape beginning at step 4.

The Apple II is sensitive to tape volume and tone settings. If you still can't get your tape to load, set the tape volume to 3 and the tone to 7 or higher. Try to load the tape again. After each failure, increase the tape volume by ½ on the volume dial. When you find the correct setting, mark it on your cassette recorder and on the tape label.

A second copy of the program is recorded on the other side of the tape. If you can't get the tape to load, reverse it and begin again with step 4.

DUMP MEMORY

*Ø.F(cr) normal hex dump
*Ø.F(ctrl-Y)A(cr) ASCII(alphanumeric) dump
*Ø.F(ctrl-Y)B(cr) binary dump
*Ø.F(sp)Ø.F(ctrl-Y)B(cr) hex & binary dump of same range
*Ø.F(ctrl-Y)B(sp)1Ø.1F(ctrl-Y)A(cr) binary & ASCII dumps of two ranges

The first command line does the normal hex dump with the Apple monitor. The second command line does a dump of a memory range in alphanumeric characters. This format can be quite useful for finding command strings or messages in a section of code that doesn't make sense during a disassembly. Often, commands or messages can be easily changed by modifying an ASCII string. The ASCII dump utility will help you find them. The third command line does a binary dump of a memory range. The fourth and fifth lines illustrate how commands can be intermixed on the same line.

ASCII DUMP FORMAT

Each line of the ASCII dump starts with the hex address of the beginning address of that line of locations and is followed by two groups of 16 characters each. The space separating the two groups is provided to make address calculation of individual characters easier. The routine works by looking at each memory location and ORAing the contents with $8Ø. This sets the high bit of the byte so it will display normally with the Apple output routines. (Note that this also guarantees that both "negative" and "positive" ASCII will be caught.) The resulting value is then checked to see if it is a control-character. If not, the byte is output. If it is a control-character, an underline character is output.

BINARY DUMP FORMAT

Each line of output contains the address of the byte and the binary representation of its value. This format can be quite useful in checking status bytes and/or ports.

CHECKSUM

*Ø.F(ctrl-Y)C(cr)

This routine computes a single byte checksum of all the bytes in the specified range and displays the resulting value. It does this by XORing (exclusive-oring) all the bytes together. This will provide a simple indication of a change in the contents of a memory range if the checksum is taken before running or moving some instructions and then again afterwards.

This routine can also be used to provide a check on the hand loading of a memory range if the correct checksum of the range is provided by the author.

For example, the checksum of the Monitor Extender ($8ØØ.CFF) is $F8.

*Ø.F(ctrl-Y)C(cr)

Now, lets change a byte:

*5:44(cr) or any byte or value in the memory range and take a look at the result:

*Ø.F(ctrl-Y)C(cr)

FILL MEMORY

*Ø.F(cr)
*ØØ<Ø.F(ctrl-Y)F(cr)
*Ø.F(cr)
*43<Ø.F(ctrl-Y)F(cr)
*Ø.F(cr)

The Fill command puts the hex value to the left of the arrow into the memory range specified on the right side.

With this command, a very large (or small) chunk of memory can be preset to an initial value. It is also possible to do this with the normal Apple monitor commands but not as conveniently. For example, here is how the second command above must be done without Monitor Extender:

*Ø:Ø
*1<Ø.EM(cr)

Another example:

*A0<400.7FF(ctrl-Y)F(cr)

This command fills the screen memory with the ASCII code for blanks.

MOVE MEMORY

*0<1.F(ctrl-Y)M(cr)
*1<0.E(ctrl-Y)M(cr)
*ADDR3<ADDR1.ADDR2(ctrl-Y)M(cr)

This will move the contents of the memory range from ADDR1 to ADDR2 to start at ADDR3. The move can be either upwards or downwards. The target location, of course, should be existing RAM memory.

Quite often a small system user encounters the problem of having enough room for code or data but in the wrong place. So, something has to be moved. The Apple monitor provides just the function — as long as you are moving the bytes downward in memory or to non-overlapping areas. But what if you want to insert three bytes of code into the middle of a routine? You have to move the top chunk far away somewhere (maybe to where some other code is residing if you've got a small system), add the new bytes, and then move the other chunk back down to where it now belongs. Or maybe a whole data area should start 37 bytes higher (for some undoubtedly arcane but essential reason)? Now that no longer needs to be a major, multi-step operation. The Monitor Extender Move command is just what the system doctor ordered.

Try the following example:

*0.F(cr) (see what's already there)
*2<0.F(ctrl-Y)M(cr)
*0.11(cr) (now look at it again)

One word of caution is in order about using this command. When moving memory upwards, remember that you may be overwriting the bytes following ADDR2, so make sure that there is enough room for the moved code to fit without clobbering other data or program areas.

STORE ASCII

MODE 1: *(address)(ctrl-Y)"TEXT FOLLOWS QUOTE(cr)
MODE 2: *(address)(ctrl-Y)"TEXT FOLLOWS QUOTE(ctrl-@)(cr)
MODE 3: *(address)(ctrl-Y)"TEXT FOLLOWS QUOTE(ctrl-Z)(cr)

The (") command allows the user to directly enter ASCII values from the keyboard into the memory locations.

In all three modes, the ASCII values of the text string following the " are entered into memory starting at (address). In mode 1, the (cr) is stored in memory at the end of the string. In mode 2, a null byte ($00) is stored at the end of the string: the (cr) is discarded. In mode 3, the routine exits after storing the last value of the string and does not enter a terminator. In all cases, the address pointer points to the next available byte so long strings may be entered in shorter segments. For example:

*300(ctrl-Y)"TEXT 1(cr)
*(ctrl-Y)"TEXT 2(ctrl-@) (cr)
*(ctrl-Y)"TEXT 3(ctrl-Z) (cr)
*(ctrl-Y)"TEXT 4(ctrl-Z) (cr)
*(ctrl-Y)"END OF TEXT(cr)

Now, take a look at the memory range just filled with either a hex or ASCII dump command:

*300.324(cr) or
*300.324(ctrl-Y)A(cr)

SEARCH MEMORY

*ADDR3< ADDR1.ADR2(ctrl-Y)S(cr)

ADDR1 and ADDR2 specify the range of memory that is to be searched. Contained at ADDR3 is the specification of the byte string for which you are searching. The first byte at ADDR3 contains the number of following bytes that are to be used in the match process. The actual search bytes start at ADDR3 + 1. For example:

*0:3 20 ED FD

This says that we are looking for three sequential bytes that contain the values $20, $ED, $FD. Any number of bytes in the range of 1 to 63 can be searched for. In this case, we are looking for a JSR $FDED instruction. (This happens to be the way most programs call for output).

When debugging or disassembling some code, it can be nice to find out where in memory a certain value or sequence of values are located. The Monitor Extender search command is designed to make the task much easier by letting the computer do most the work. The command syntax is not quite as simple as it could be, but allows for greater flexibility in use.

If we type in the following command, we will discover every location in the monitor ($F800-$FFFF) that calls for output.

*0<F800.FFFF(ctrl-Y)S(cr)

If the monitor prompt (*) returns without outputting anything else, the byte sequence was not found in the specified range. If it was found, the starting address(es) of the string location(s) will be output and then the prompt will reappear.

Specifying a string length of zero will result in an error message. A string length of 64 or more is masked to be in the range 0-63. Results will be unpredictable at best if the I/O address range is searched because accessing some of these locations controls such things as text/graphics, paddles, screen1/screen2, etc. With some care, the I/O slot ROMS can be accessed but be very sure of which addresses you are looking at (see the Apple Reference Manuals for more detailed information).

This command can be quite powerful if used in conjunction with the ENTER ASCII command to build the strings, but bear in mind that the text entered that way is "negative" ASCII and will not match "positive" ASCII strings in memory. Also, many string storage techniques will set or clear the high bit of the first or last character of the string rather than use explicit delimiters; this might result in not finding the string because an exact match (of the exact length) is required.

THE DISASSEMBLER

*ADDR1.ADDR2(ctrl-Y)Z(cr)

The "Z" command disassembles a range of addresses, creates a labelled text file in memory, and then checks for references to the labels within the file, deleting any labels not found. This routine is designed to be general purpose and is not tied to any particular editor and/or assembler. It does not, therefore, generate anything more than a slightly modified text file of the disassembly procedure. In most cases, varying amounts of editing will have to be done to make the file compatible with your assembler. It does, however, provide the basic text from which you can make whatever modifications you desire.

Any previously existing text in the buffer area is lost with this command. Also, before using this command, make sure that the text file pointers in page zero have been set up. Restarting the Monitor extender will do this for you:

*800G

The disassembler is two-pass in operation. Each pass is discussed in detail below.

During pass 1, the memory range is disassembled using the monitor's disassembler. As each line is written to the screen, another line buffer (at locations $2D0-$2FF) is being filled with a modified line. This line is of the following general format:

ZADDR(sp)OPC(sp)OPERAND(sp)(cr)

Some examples:

Z1234 LDA Z45	page zero
Z1235 STA Z1365,X	absolute, indexed
Z1236 DEX	implied
Z1237 LDX #$33	immediate
Z1238 DFB $FF	illegal (define byte)

After each line has been created in the buffer, it is moved into the text buffer area defined by bytes $A-$F in page zero. This process continues until either the memory range is complete or until the buffer is full.

At this point, the screen will stop scrolling, the bell will ring, and pass 2 begins.

During pass 2, the routine moves through the file label by label checking for references to the label. If the label is found elsewhere in the file, it is retained; if not, the label is deleted by replacing it with blanks. The blinking square in the upper right corner of the screen is an indicator that the routine is working and has not gone off into the ozone somewhere. It changes state as each new label is looked for. The speed of the blinking will give you a rough idea of the size of the file — the faster the blinking, the fewer the number of labels, i.e., the shorter the file.

PLEASE NOTE

A full 7K buffer file takes 1½ to 3 minutes to disassemble and process. Approximately 400-500 text lines (and labels) are generated with a file this size (depending, of course, on the nature of the code). This represents about ¾ K of code.

By changing the size of the buffer, of course, correspondingly larger or smaller programs can be disassembled. Be aware, though, that the larger the file, the more label checking that has to be done.

Pass 2 ends with the bell again and the monitor prompt (*) returns to indicate the user is now back in control. After this pass, any lines with labels that could not be found in the text file will have been changed as follows:

```
Z1234 LDA Z5678      after pass 1
      LDA Z5678      after pass 2
```

Pass 2 can be halted at any time by pressing "RESET". The file will (probably) have been only partially "delabelled" but it can still be listed, saved, or whatever. The delabelling procedure can be restarted by the command:

```
   *XYZG          where XYZ is Monitor Extender's starting address + $35B
ex: *B5BG         for Monitor Extender residing at $800
```


THE + COMMAND

This command is of the same format and function as the "Z" command except that the text file is not reset first. It allows the user to append one disassembly (or more) to the end of existing ones up to the maximum size of the text buffer. The pass 2 procedure always starts at the beginning of the whole file. If, however, a label that is referenced in a later disassembly is not referenced in an earlier one, it will have already been blanked out by the early passes.

To conserve space, the "<" command can be used between subsequent disassemblies without any ill effects. Please see the section on text file commands about its usage.

TEXT FILE COMMANDS

LISTING THE TEXT FILE

*(ctrl-Y)L(cr)

This command lists the text file that resides between the addresses specified in locations $A,B and $E,F In most cases, this file will have been created with the disassembly command ("Z" or " + "). Each line of text is preceded by a decimal line number, but this number is NOT a part of the file. The listing routine generates the line numbers for display only. At the end of the file listing, the beginning address of the file and the next available byte after the file are printed. For example,

C00-19A7

Location $C00 is the first byte of the file and location $19A6 is the last byte of the file.

COMPRESSING THE TEXT FILE

*(ctrl-Y)<(cr)

This command compresses the text file by replacing all multiple blanks with a single blank. This command was implemented to allow small memory users to utilize as much of their machines as possible by eliminating (possibly unneeded) redundancy . When a file is created with the disassembler, a pseudo-label is created for each line of code. The dissassembler then checks to see if that label

is referenced anywhere in the file. If not, it is eliminated by replacing it with blanks. Some assemblers that might then work on the created file have very strict requirements about where in the source line opcodes must begin; others will allow a more free-form style.The "<" command allows the user of a looser format assembler to create larger text files in a given size of memory. The result, however, is a little harder to read and most users might choose to retain the unaltered file. (For more information, please see the section regarding the disassembler commands.)

THE ? COMMAND

*(ctrl-Y)?(cr)

This command prints out the text file addresses without listing the file.

UTILITY COMMANDS

THE "&" COMMAND

*(ctrl-Y) & (cr)

Since Monitor Extender uses the CONTROL-Y vector to hook into the monitor, it is no longer possible to call user programs this way without losing easy access to Monitor Extender. Provision has been made for this contingency by setting up a new command that does the same thing. The above command will call a sub-routine beginning at address $3F5.

Normally, this location will contain a JMP instruction to the actual beginning of the routine. When Monitor Extender is initialized, it sets up this vector to point to the monitor ($FF65). This is not very useful but at least you know what will happen should you inadvertently give the command—nothing. A more constructive use would be to have it point at the SLOLIST address of Monitor Extender since pressing "RESET" will always disconnect that routine.

(This vector is also used by APPLESORT II; if a "&" is the first character in a program line, it will do a JSR to $3F5. The Integer Basic ROMs also contain some floating point math utility routines that use this vector as the error exit, so some care is necessary if the Monitor Extender is to be used simultaneously with them).

SLOLIST

Included in the code for Monitor Extender is a routine that provides the user with some of the capabilities of the AUTOSTART ROM. By executing a call to $8Ø6, a routine is hooked into the output chain that allows the user to start, stop, or abort a listing on the screen. The routine also slows down the speed at which lines are output. The SLOLIST feature looks at every character that is sent to the output.

If it is a carriage-return ($8D, it delays slightly and then looks to see if the space bar was pressed. If the space bar was pressed, the routine goes into a loop waiting for another key to be pressed. If any key except the space bar is now pressed, normal output continues. Pressing the space bar, however, puts out just the next line of text. This allows you to step through a listing one line at a time. Entering a (ctrl-C) aborts the listing (or whatever) by jumping to the address contained in the SOFTEV (Soft Entry Vector — locations $3F2 and $3F3). Monitor Extender normally sets this up to point to the monitor; you could, however, have it point to the warmstart entry of BASIC for use while running BASIC programs (for ROM versions this is at $EØØ3).

Pressing "RESET" will always disconnect this routine so it might be useful to set the "&" vector (at $3F5) to point to address $8Ø6. This allows easy reconnection of the SLOLIST feature. The machine takes care of modifying the CSWL locations as needed (it sets locations $36 and $37 to the actual address of the SLOLIST routine -$863).

This feature works with all of Monitor Extender (including the disassembler) and is quite handy when scanning large dumps of any kind.

DISK COMMANDS

The Monitor Extender also contains commands that allow the machine language programmer to access any track and sector on a disk without the Disk Operating System getting in the way. A portion of the DOS called the "RWTS subroutine" (Read Write Track Sector subroutine) is used to provide this capability. Because of the complexity of the DOS, the user is encouraged to read chapter 9 of the Apple DOS manual for an overview of what the RWTS subroutine does and how it is used. In addition, Appendix C of the DOS manual gives a very complete explanation of the physical layout of information on an Apple II diskette.

The following information is mainly about how the Monitor Extender uses the RWTS subroutine to provide the user with direct access capabilities. Some suggestions and examples will be given to show practical uses but this is not intended to be a tutorial on how the DOS works or how to handle all possible situations.

General Format of Disk Access Commands

The format of the disk access commands is similar to the other commands of the Monitor Extender with the addition of one more character. After the (ctrl-Y) comes a "/" to indicate a disk command. This allows some command letters to be reused when a particular character seems the best to use for that type of command (the "?", for instance); it also forces the user to do a little extra typing. In most situations, this would not be a desirable goal but with these commands we are dealing at a very basic — almost hardware — level with a very complex piece of equipment and the software that drives it. We no longer have the insulating buffer of the disk operating system to protect our information from silly errors and slips of the fingers. If the slight extra effort saves a valuable file or disk, so much the better.

In general, then, the format for disk access commands is this:

PARAMETER1.PARAMETER2(ctrl-Y)/COMMAND(cr)

Initializing For Disk Commands

****

THE FOLLOWING COMMAND MUST BE GIVEN BEFORE ANY OF THE MONITOR EXTENDER DOS COMMANDS ARE USED:

****

*(ctrl-Y)/I(cr)

This sets up a disk I/O block (DIOB) and a data buffer for the RWTS subroutine to use. If this is not done, chaos will probably result. While learning how to use these commands, it is a good idea to use a disk that has been initialized but does not contain any valuable information. If something goes wrong, then, nothing important will have been messed up.

The DIOB is set up at address $300 with the following default values (assuming the Monitor Extender is running from $800):

ADDRESS	VALUE	EXPLANATION
	DISK I/O BLOCK	
$300	$01	type of IOB
$301	$60	slot number times 16
$302	$01	drive number
$303	$00	expected volume number
$304	$00	track number
$305	$00	sector number
$306	$11	address of DCT lo
$307	$03	address of DCT hi
$308	$00	address of data buffer lo
$309	$0D	address of data buffer hi
$30A	$00	unused
$30B	$00	unused
$30C	$00	command code
$30D	$00	error code
$30E	$00	actual found volume number
$30F	$60	previous slot number times 16
$310	$01	previous drive number
	DEVICE CHARACTERISTICS TABLE	
$311	$00	device type
$312	$01	number of phases
$313	$EF	time count lo
$314	$D8	time count hi

The data buffer is 256 bytes long and is located in the page of memory between the end of the Monitor Extender code and the beginning of the text file buffer. If you want to move this buffer, change the values of the DIOB to point to the beginning address of whatever area of memory you want to use; the buffer need not start on a page boundary. (Rather than moving information in or out of the buffer to memory you can just change the buffer address to point at different sections of memory. Remember, though, that 256 bytes will always be read from or written to the buffer.)

You will have noticed after executing the initializing command that a line of inverse video information was printed before returning to the input mode. Every disk command in the Monitor Extender ends with the printing of this status line.

Disk I/O Status

Upon completion of a disk command, the Monitor Extender shows the current status of several locations in the DIOB. The format for this line (in inverse video) is:

SLOT# DRIVE# TRACK# SECTOR# BUFFER ADDRESS

If an error has occurred in a disk access, the status line will be proceeded by a line in inverse video that reads:

ERR XX

The possible error codes are:

10 — diskette is write protected
20 — volume mismatch error
40 — drive error
80 — read error

Some possible error reasons:

10 — the diskette has no write protect notch
— the write protect notch has been covered
— the diskette is inserted incorrectly

20 — the DIOB expected volume number does not match the actual volume number found (location $3Ø3 does not equal location $30E)

40 — a track number greater than 34 ($22) was given
— a sector number greater than 12 ($ØC) was given

80 — the sector was never written to, hence, cannot be read
— cannot find that track and/or sector

If an error occurs, the error code will be found in the DIOB at location $3ØD. If there is no error, this location is NOT reset to zero but contains (at least when reading) the last data byte read in. In other words, it is a valid error code only if an error occurred. Errors are indicated by the carry flag being set upon return from the RWTS subroutine.

In most cases, the contents of the data buffer are not altered when an error occurs unless that error involves a checksum error while reading data.

(The examples given in the DOS manual on pages 95 & 96 are incorrect in that no check is made for errors before returning. There is no way to check the carry flag from Basic so it must be done in machine language if it is to be done at all. The code at $CF4 in the Monitor Extender shows one way of handling the situation.)

The status line can be obtained at any time by typing

*(ctrl-Y)/?(cr)

SETTING SLOT AND DRIVE

*X.Y(ctrl-Y)/#(cr)

This command sets the new slot number to X and the new drive number to Y. The initialization procedure defaults to slot 6, drive 1 but that may not be where your target drive is. This command allows you to change it as you need.

The "/#" routine first sets the previous slot and drive numbers from the current numbers and then sets the new slot and drive from the command line. Note that you do not have to specify the slot number times 16, just the slot number. Some masking is done so that the slot numbers can range only from $Ø to $7 and the drive numbers from $Ø to $3. This doesn't catch all possible errors but does cut them down.

The RWTS subroutine checks the previous slot and drive (if they are different from the target slot and drive) to make sure it has stopped before attempting to access the new one. This is why the previous drive has to be specified. By using the "/#" command twice in a row, you set a "new" drive that is also the "old" drive.

*(ctrl-Y)/I(cr)	initializes to slot 6, drive 1
*5.2(ctrl-Y)/#(cr)	sets slot 5, drive 2 but will check slot 6, drive 1 before accessing the new drive.
*5.2(ctrl-Y)/#(cr)	sets new to slot 5, drive 2 and will check slot 5, drive 2 as old drive also.

Asking for the DIOB status will always give you the current slot and drive numbers but you have to check locations $3ØF and $31Ø or use the above procedure to be sure of the previous values.

There will probably be times, of course, when you will be working with two or more drives at once. In that case, just changing the slot and drive numbers (once) back and forth as needed is the correct procedure.

READING A SECTOR FROM DISK

*TRACK.SECTOR(ctrl-Y)/R(cr)

Once a drive has been selected, an individual sector is read into the disk data buffer with this command.

The track number must be in the range $00-$22 and the sector number must be in the range $0-$C or errors will result. If a sector has never had any information written to it, a read error will occur. A normal return will be signalled by the printing of the status line and the monitor prompt.

If an error situation arises, the disk may make funny noises or seem to take an awful long time to return to the user prompt; this is because the RWTS subroutine attempts to do its job many times before it finally gives up and returns waving the carry flag.

Upon normal completion of a read, the sector's information will be in the data buffer that starts at the address on the right end of the status line. This information can now be examined or modified with the normal or extended monitor commands.

WRITING THE DATA BUFFER TO A DISK SECTOR

*TRACK.SECTOR(ctrl-Y)/W(cr)

The data buffer can be filled with any 256 bytes of information that you wish; it is written out to a disk sector with this command. Upon completion of the writing, the status line and the monitor prompt will return indicating you can continue.

The normal use of this command comes after a sector has been read in and a byte (or bytes) modified. The sector is then overwritten with the modified data. Note, however, that since you specify where the information is to go, you can write it to a different sector (or a different disk or a combination of the two) than it came from.

Since the rewriting of a sector is the most common mode of operation, a command exists to write the buffer to whatever track and sector values exist in the DIOB. Usually, the last disk command was a read so these values will reflect where the information in the buffer came from. (This information is always displayed in the status line). To rewrite the sector, you just type:

*(ctrl-Y)/>(cr)

READABILITY VERSUS WRITEABILITY

Although a sector cannot be read if it has not been written to, you can write to any sector regardless of its previous status. When a disk is physically formatted, just the volume, track, and sector numbers are written on the disk in addition to what can be called clock pulses. The space is left for data but it is not actually written. Initializing a disk under DOS does all the physical formatting plus the writing of DOS itself and the needed housekeeping information such as the directory and the VTOC (Volume Table Of Contents - the track bit map, etc. See the DOS manual for a complete explanation).

PHYSICALLY FORMATTING A DISK

****

THIS COMMAND SHOULD BE USED WITH EXTREME CARE AS IT WILL DESTROY ANY INFORMATION EXISTING ON A DISK!!!!

****

The format command allows you to create a disk that is physically useable under DOS but with absolutely no information on it. The physical formatting of tracks and sectors is done but that is all — no DOS, no directory, no VTOC — nothing!

This capability is probably useful only to programmers who want to use the RWTS subroutine to write their own operating systems or virtual memory storage systems but is included here for completeness. The format for the command is:

*VOLNUMBER(ctrl-Y)/F(cr)

VOLNUMBER must be in the range of $1 to $FE to be compatible with the Apple DOS. If it is omitted, the RWTS subroutine will pick up the current volume number from the DIOB as the value to use. (The default value in the DIOB is $00 so it will match any volume number of the RWTS subroutine finds in its other functions.)

The only way to change the volume number of a disk is to reformat it or reinitialize it since this number is stored as part of every track and sector address mark on the disk (these marks are inaccessable to the user).

Even though a volume number cannot be changed, many other things can be done with a disk at this level of access. This brings us to our next topic — some applications of the routines and information.

SOME THINGS TO TRY WITH YOUR DISK

1. MASTER/SLAVE

To determine whether a given disk is a master (relocating DOS) or a slave, read in track $Ø, sector $1 to the buffer.

```
*Ø.1(ctrl-Y)/R(cr)
```

Examine the buffer with the monitor's "L" command:

```
*DØØL(cr)
```

If the first instruction is "STX$37E9", the disk is either a master or a 16K slave. If the instruction is "STX $77E9", it is a 32K slave; if it is "STX$B7E9" the disk is a 48K slave.

If it might be a master ("STX $37E9"), load track $Ø, sector $A into the buffer:

```
*Ø.A(ctrl-Y)/R(cr)
```

and list it:

```
*DØØL(cr)
```

If the first two instructions are:

```
JMP $1D84
LDA #$BF
```

the disk is a master disk.

2. EXAMINING AND MODIFYING THE DIRECTORY

The directory for each disk begins in track $11, sector $C and extends down through track $11, sector $1 as needed. By loading the sectors into the buffer and using the various monitor and Monitor Extender commands, you can rename files, unlock or lock files, find control characters embedded in file names, etc.

By tracing the track/sector linking lists and resetting the appropriate bits in the VTOC, it is even possible to resurrect a file that was inadvertently deleted. This a complex process that is best understood by reading appendix C of the DOS manual. It can only be done if no new information was written to the disk after the file was deleted since the new information may have overwritten the directory information or some of the file's sectors.

3. CHANGING THE NAME OF THE BOOT PROGRAM

Whan a disk is initialized, the name of the greeting program is placed in the DOS image on the disk. Every time DOS is booted it attempts to run a program by that name. The DOS manual gives a procedure for changing that name but it can be done only when a disk is updated from a slave to a master. If you want to keep the disk a slave, you've got problems — until now.

Track $1, sector $A of a slave disk contains the name of the greeting program starting at relative location $75 within the sector. (The name is in track $1, sector $C of a master disk.)

With the (ctrl-Y)" command of the Monitor Extender, you can place your new name into the buffer (padded with blanks as needed to fill 3Ø bytes) and rewrite the sector. If the original name was "GREETINGS", you could change it as follows:

```
*1.A(ctrl-Y)/R(cr)
*DØØ.DFF(ctrl-Y)A(cr) make sure you got it
*D75(ctrl-Y)" HELLO(sp)(sp)(sp)(sp)(ctrl-Z)(cr)
*DØØ.DFF(ctrl-Y)A(cr) check for extraneous characters
```

If it's o.k.:

```
*(ctrl-Y)/>(cr) store it back
```

Now, whenever the disk boots, it will try to run a program called "HELLO" instead of "GREETINGS"; you should, of course, have such a program on the disk to avoid a "FILE NOT FOUND" message.

USING DOS FROM THE MONITOR

If DOS has been booted, it can be used with the monitor or the Monitor Extender by making sure it is hooked into the I/O hooks with this command:

*3EAG(cr)

Now, all the normal DOS commands are available just like in Basic. You can CATALOG, LOAD, BLOAD, RUN, BRUN, etc. to your heart's content.

To use the SLOLIST feature of the Monitor Extender, you type:
*806G(sp)3EAG(cr)
This hooks up the SLOLIST and then rehooks DOS with the SLOLIST address.

MOVING THE MONITOR EXTENDER

When Monitor Extender is initially loaded with the instructions contained in the beginning of this documentation, two different things are actually being loaded. One, the screen message containing the copyright notice, title, and other stuff; and two, the actual code for Monitor Extender. The code itself occupies the memory block from $800 to $CFF. The user should save just this block onto either tape or disk so it can be easily loaded into other memory ranges. The code will run from A-N-Y place in memory as long as the first instruction starts on page boundary ($XX00). To use Monitor Extender somewhere else, just start it with the command:

*XX00G (where "XX" is the starting page)

The initialization figures out where it is running from and sets up the appropriate vectors. The Cold Start ($XX00G) sets the text buffer to start 256 bytes above the end of the code of the Monitor Extender and to extend through $1B pages of memory.Thus,the Monitor Extender sets itself up at the bottom of a 8.5K memory block with the text buffer in the top 7K. If you are not going to be using the text buffer functions, you could move the Monitor Extender to sit up in the top 1.5k of available RAM. The buffer of 265 bytes between the end of the Monitor Extender code and beginning of the text buffer is used for the disk access buffer.

By entering Monitor Extender at the Warm Start point ($XX03G), the same initialization is done except that nothing is done to the text pointers in page zero.(A Cold Start resets the pointers so Monitor Extender thinks there is no text yet.)

It is suggested that you don't try moving Monitor Extender with the extended move command as trying to move code that is executing can be hazardous to your sanity and the integrity of the code. Use the normal Move command as needed to position the code in a new environment.

A note for the advanced programmer

A major goal in the coding of Monitor Extender was to make it position independent; as a result, if you examine the code you will discover various types of inefficiencies and unusual branching statements. This is the price one pays for relocatability with a machine that is not designed for it. In short, Monitor Extender is not a sterling example of clear programming, but it does strike a useful balance between the competing factors.

SAVING THE MONITOR EXTENDER ON DISK

Boot up the disk. From Basic, enter the monitor with the following command:

>CALL-151 (or]CALL-151)

You can now load the Monitor Extender tape as explained earlier. After the Monitor Extender has been loaded, it can be saved with the following sequence of commands:

*3EAG this makes sure DOS is connected
*BSAVE MONITOR EXTENDER,A$800,L$500(cr)
*CATALOG(cr) make sure it got saved

The Monitor Extender can now be loaded or run from DOS with either:

>BLOAD MONITOR EXTENDER (to run, type "CALL 2048" from Basic or "800G" from the monitor)

or

>BRUN MONITOR EXTENDER (loads and runs)

As explained in the DOS manual, the program can be loaded and run from a different location. For example:

>BRUN MONITOR EXTENDER,A$4000(cr)

will load the program into memory at address $4000 (decimal address 16384) and run it from there. Remember that the Monitor Extender must run from a page boundary.

MONITOR EXTENDER INTERNALS

PAGE ZERO LOCATIONS USED

Monitor Extender makes extensive use of the monitor locations A1L—A5H ($3C-$45) for many of the same things the Apple monitor uses them for—addresses and data bytes. In addition, locations $A-F are used as the text file pointers and are used by the disassembler, the (ctrl-Y)L command, the < command, and the ? command.

Location	$A	text begin lo byte
	$B	text begin hi byte
	$C	maximum text page lo byte
	$D	maximum text page hi byte
	$E	next available byte for text lo byte
	$F	next available byte for text hi byte

OTHER LOCATIONS USED

The disassembler uses a temporary line buffer at $2D0-$2FF to build up the line it is going to place into the text file. The disassembler uses the screen memory as generated with the monitor's disassembly routine to pick up the text it is going to manipulate, so the "output switch" (CSWL at locations $36-37) M-U-S-T eventually point to the video driver routine in the monitor ($FDF0). The following command will set this up if needed:

*0(ctrl-P)(cr) (or press "RESET")

The following locations in page 3 are set up by the initialization section of the Monitor Extender code:

LOCATION		VALUE	
SOFTEV	$3F2	$65	
	$3F3	$FF	
PWREDUP	$3F4	$5A	(EX-OR of $A5 and the byte in $3F3)
AMPERV	$3F5	$4C	
	$3F6	$65	
	$3F7	$FF	
CNTRLYV	$3F8	$4C	
	$3F9	$89	
	$3FA	$08	

The values for the CNTRLYVector are for Monitor Extender running at $800.

The following is a table of the various entry points in the Monitor Extender code for the routines.

(All addresses are given for Monitor Extender running from $800.)

$800	COLDSTART	$809		
$803	WARMSTART	$823		
$806	SETSLOLIST	$852		
	SLOLIST	$863		
	SEARCH	$894		
	&	$8CA		
	ASCIIDUMP	$8E2		
	MOVE	$901	(BACKTOMON)	($BDD)
	"	$95F	INITDISKIO	$BEF
	FILL	$989	SLOT/DRIVE	$C3A
	CHECKSUM	$999	TR/SEC READ	$C60
	LIST	$9B8	TR/SEC WRITE	$C77
	FILESIZE	$A14	FORMATDISK	$C7F
	BINARYDUMP	$A2E	REWRITEBUFF	$C8F
	<	$A59	(CALL RWTS)	$C94
	Z-disassm	$AA0	IOBSTATUS	$CA3
	+-disassm	$AA8	(DISKERRORS)	$CE9
	(DELABEL	($B5B)		

The entry point from the (ctrl-Y) JSR is to location $889. The initial addresses for the text file are set as follows by the COLDSTART:

LOCATION	VALUE	
$ØA	$ØØ	
$ØB	$ØE	(start-page + $Ø6)
$ØC	$ØØ	
$ØD	$29	(Loc $ØB + $1B)
$ØE	$ØØ	
$ØF	$ØE	(= loc $ØB)

The values in locations $ØB, $ØD, and $ØF may be changed by altering the immediate value bytes in the instructions at locations $811 and $817. The ADC (add with carry) instructions do not check for carry afterwords, so by using large values, it is possible to set the text file below Monitor Extender. (That is, the values "wrap around".) For example, if Monitor Extender has been moved to start at location $4ØØØ, adding $FØ in the instruction at $4Ø11 ($811) will set the starting address to $3ØØØ. ($4Ø + $FØ = $3Ø with carry set.)

In a like manner, the size of the text file can be altered by changing the byte in the instruction at $817.

The "Z" command resets locations $ØE and $ØF to equal locations $ØA and $ØB. The "+" command does not alter the text file pointers before beginning to disassemble.

If the instruction at location $B58 is changed from a JSR to a JMP, then pass 2 of the disassembler is disabled. This could be useful if you wanted to create and append several files and then do the delabelling. The delabel routine will then have to be called manually (or you could set the "&" vector to point to it).

ABOUT THE CONTROL-Y COMMAND

When the monitor encounters a (ctrl-Y) in the keyboard input stream, it does a JSR to location $3F8. If the command is of the following format,

*ADDR4<ADDR1.ADDR2(ctrl-Y)

the value of ADDR4 is placed in locations $42 and $43, the value of ADDR1 goes into locations $3C and $3D, and the value of ADDR2 is put into location $3E and $3F. The JSR to location $3F8 is then executed. Location $34 contains the offset into the keyboard buffer (locations $2ØØ-$2FF) of the next character after the (ctrl-Y).

COMMAND SUMMARY

Monitor Extender Only

Like most of the APPLE Monitor commands, some Monitor Extender commands may take long or short versions. The commands can also be intermixed on the same line. In the following summary, those commands that can use the alternate forms listed below are marked with an asterisk (*).

RANGE	ACTION
ADDR	executes for range of 1 byte
ADDR1.ADDR2	executes for range of (?) bytes
.ADDR	executes from last address of last command to new address.

ASCII dump *	ADDR1.ADDR2(ctrl-Y)A
BINARY dump *	ADDR1.ADDR2(ctrl-Y)B
CHECKSUM	ADDR1.ADDR2(ctrl-Y)C
FILL memory	VALUE<ADDR1.ADDR2(ctrl-Y)F
LIST text	(ctrl-Y)L
MOVE memory	NEWBEG<OLDBEG.OLDEND(ctrl-Y)M
SEARCH memory	PATTERNADR<ADDR1.ADDR2(ctrl-Y)S
DISASSEMBLE/NEW	ADDR1.ADDR2(ctrl-Y)Z
DISASSEMBLE/APPEND	ADDR1.ADDR2(ctrl-y)+
COMPRESS text file	(ctrl-Y)<
SIZE of text file	(ctrl-Y)?
USER subroutine	(ctrl-Y)&
ENTER ASCII	ADDR(ctrl-Y)"text follows
" "	(ctrl-Y)"part 2
INITIALIZE DISK I/O	(ctrl-Y)/I
GET DISK IOB STATUS	(ctrl-Y)/?
SET SLOT & DRIVE	SLOT.DRIVE(ctrl-Y)/#
READ TRACK & SECTOR	TRACK.SECTOR(ctrl-Y)/R
WRITE TRACK & SECTOR	TRACK.SECTOR(ctrl-Y)/W
REWRITE TRACK/SECTOR	(ctrl-Y)/>
FORMAT DISK	VOLNO(ctrl-Y)/F

CARE AND HANDLING

Keep the enclosed magnetic recording away from magnetic fields, transformers, power supplies, motors, etc. so that the program will not be erased. Always protect magnetic cassettes from temperature and humidity extremes.

COPYRIGHT

LIMITED WARRANTY

IMAGE Computer Products, Inc. shall have no liability or responsibility to purchaser or any other person or entity with respect to any liability, loss or damage caused or alleged to be caused directly or indirectly by this product, including but not limited to any interruption of service, loss of business and anticipatory profits or consequential damages resulting from the use or operation of this product. This product will be exchanged if defective in manufacture, labeling or packaging, but except for such replacement the sale or subsequent use of this program material is without warranty or liability.

IMAGE
COMPUTER PRODUCTS, INC

MADE IN U.S.A. FORM: ICP-2006 PRINTED IN U.S.A.

IMAGE
COMPUTER PRODUCTS

APPLE II
16K, CASSETTE

Monitor Extender™ 2006

This utility program works in complete harmony with the Apple monitor to extend your computer's capability and help you use the full power of machine language programming.

Screen display shows memory in HEX, ASCII or BINARY. Move data anywhere in memory without regard to direction or overlapping and read or write any sector on disk. Insertions may be in HEX or ASCII so you can easily format high speed text displays without conversions.

Study, modify or disassemble any program, complete with labels. Several programs may be combined, and the entire disassembled text file stored on disk/tape for later assembly. The slow listing feature steps through listings with ease.

THIS CASSETTE PROGRAM REQUIRES A CASSETTE PLAYER, 16K MEMORY.