cassette tape operating system

Datapoint

Datapoint 2200

Cassette Tape

Operating System

May, 1972

DATAPOINT CORPORATION

4/6/72

Datapoint 2200 Cassette Tape Operating System

CASSETTE TAPE OPERATING SYSTEM

SECTION I

INTRODUCTION

1.0 THE OPERATING SYSTEM

The operating system is a conversational mode program for the Datapoint 2200 which allows the user to catalog, load, debug, run programs, and provide other utilities important to the user of the Datapoint 2200.

The operating system itself is a relatively long program which is overlayed when user programs are called in from tape. However, a family of resident utility routines is loaded with the operating system that may be used by user programs to simplify frequently used functions such as reading from keyboard, writing to CRT screen, reading and writing tape records, etc.

When power is first applied to the Datapoint 2200, the first step must be to load a block of data from the rear tape cassette deck into the processor's memory and transfer control to it. In the operating system, this first block of data is called a LOADER and when control is transferred to it, it proceeds 1) to first check itself to see if it was loaded properly 2) to determine if a version I or a version II machine is being used and then 3) to load the next file on the same tape which is the rest of the operating system program. This process can be executed at any time (assuming a proper program tape is in the rear deck) by pressing the RESTART key on the right hand side of the 2200.

This first block of data can be used to load programs other than the operating system and is generally useful for all applications of the 2200. In order to use the operating system, a full 8K bytes of memory must be provided with the 2200, but the loader can be used with any size memory.

1.1 FUNCTION

The primary function of CTOS is to provide the user with an easily accessible data environment which will greatly facilitate program generation. This function is fulfilled through the use of a file handling system which is available both directly from the keyboard in the form of system commands and through program calls to file handling input/output subroutines. Note that the keyboard facility deals mainly with the system (rear) tape (using the data [front] tape mainly for input/output and scratch space) but that the program routines are generalized to allow use of either tape.

1.2 KEYBOARD FACILITIES

The keyboard accessible facility allows the user to fetch and execute object files, which may be either system packages, such as the editor and assembler, or files the user has generated with either the assembler or other code generating programs. This facility also allows the user to create new files, alter or delete old ones, or perform certain utility functions. The system tracks the files on the system tape in a symbolic catalog which may be manipulated by the operator at the keyboard or used in program linking.

Keyboard accessible facilities also include tape positioning, listing and tape modification.

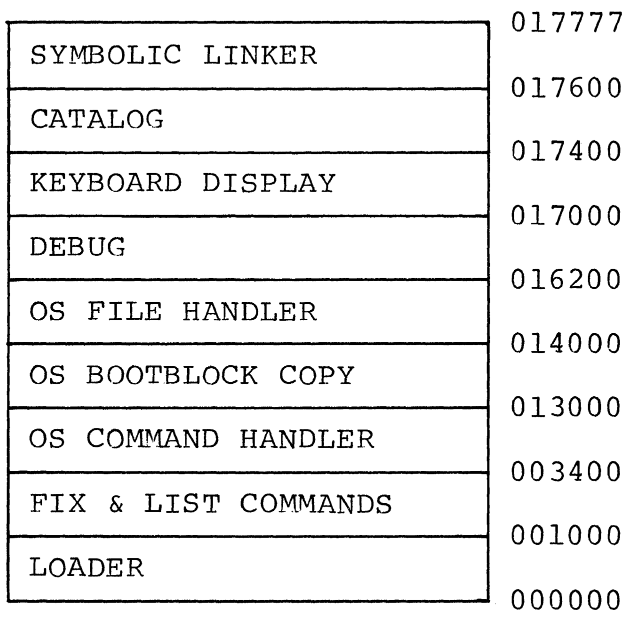
1.3 PROGRAM FACILITIES

The program routines perform basic operations such as reading and writing records with all parity checking and generation handled for the user. Other operations such as positioning to the beginning or end of a file, backspacing over records, or rewinding the tape are also provided. Parameterization is handled in a generalized way to make subroutine usage easy and consistent.

1.4 PHYSICAL LAYOUT

The memory layout of the operating system is shown below. The OS FILE HANDLER is the program accessible facility mentioned above while the OS COMMAND HANDLER is the keyboard accessible facility. Note that only 017400 and up need be in memory if only the symbol linker (which calls in an overlay by name so that its physical file number may be changed without having to rewrite the program calling in the overlay) is to be used, only 016200 and up need be in memory if only the debugging tool is to be used, and only 014000 and up need to be in memory if the keyboard facilities are not to be used (of course, 00777 is always reserved by the system).

CTOS MEMORY USAGE MAP:



CASSETTE TAPE OPERATING SYSTEM

SECTION II

TAPE FORMATS

2.0 THE LOADER

The loader is the heart of CTOS. It enables other programs to load files from the tapes into memory without the tape having to be at the beginning of the desired file and provides extensive error protection. It is the routine used by the bootstrap mechanism (indeed, it is part of the bootblock) to load the initial program and is also the routine used in overlay and linkup operations both by CTOS and utility packages.

2.1 BOOTSTRAP ACTIONS

When a restart occurs, the rear deck is rewound and the first block on the tape (called the bootblock) is loaded into memory starting at location zero. The first 512 bytes of memory (0 to 0777 octal) have been reserved for a permanently resident program which is loaded from the bootblock. The first 44 bytes of this block constitute a program which runs a parity check on the rest of the block that should have been loaded. The processor is halted (note auto-restart implications if the auto-restart tab on the cassette is punched out) if this routine finds a fault in the check. Otherwise, zeros are stored in the memory locations used in the parity check routine. This will cause a halt if an early data drop-out from the tape machine occurs during the next bootstrap load (typically only one or two bytes get loaded in this failure mode).

The loader then stores a mask into location 020663 to set the memory limit mask for a version I or a version II machine. Since memory addresses in version I (machines wrap around at 020000, the mask (037 for 8K limit) will be stored at location 0663 in the loader. In version II machines no wrap around occurs and the original mask (077 for 16K limit) will be unchanged.

2.2 FILE ORGANIZATION

Once file zero has been loaded from the system tape, the bootstrap program (locations 0 through 044) is never used again until the next restart operation which will overlay it. The loader, however, will be used many times. The physical layout of information on the system tape is as follows:

BOOTBLOCK	FILE0	FILE1	...	FILE31	FILE32	FILE127

File 0 is the one executed by the bootstrap and is typically followed by a sequential (required to be sequential by the loader) set of minimally increasing (file numbers go up by only one at a time) files up to 31 (a CTOS catalog size limitation, although the loader will load a file with any positive numbers), followed by a file 32, which is a system scratch file, followed by a file 127 (largest positive eight bit number), which is a dummy to mark the logical end of the tape.

2.3 FILE LAYOUT AND RECORD FORMAT

Each file is a group of records starting with a very special four byte record. Every record used by CTOS starts with two special bytes to indicate that it is one of three types: file marker, numeric data, or symbolic data. The file marker, which is the special four-byte record at the beginning of a file, contains two additional bytes that denote the file number. The use of two bytes for both the record type and file number provides redundancy for error control, since the second byte is simply the one's complement of the first. The record types are denoted by 0201 for file marker, 0303 for numeric data, and 0347 for symbolic data. The following tape summarizes all of the various data formats used by the system. XP and CP denote the two longitudinal parity checks and will be described later. FN denotes the file number and FN its one's complement.

FILE MARKER RECORD: 0201/0176/FN/-FN

NUMERIC DATA RECORD: 0303/074/XP/CP/DATA

SYMBOLIC DATA RECORD: 0347/030/XP/CP/DATA(with VRC)

FILE: FILE MARKER/DATA RECORD/DATA RECORD/...

SYSTEM TAPE: BOOTBLOCK/FILE0/FILE1/.../FILE31/FILE32/
FILE127

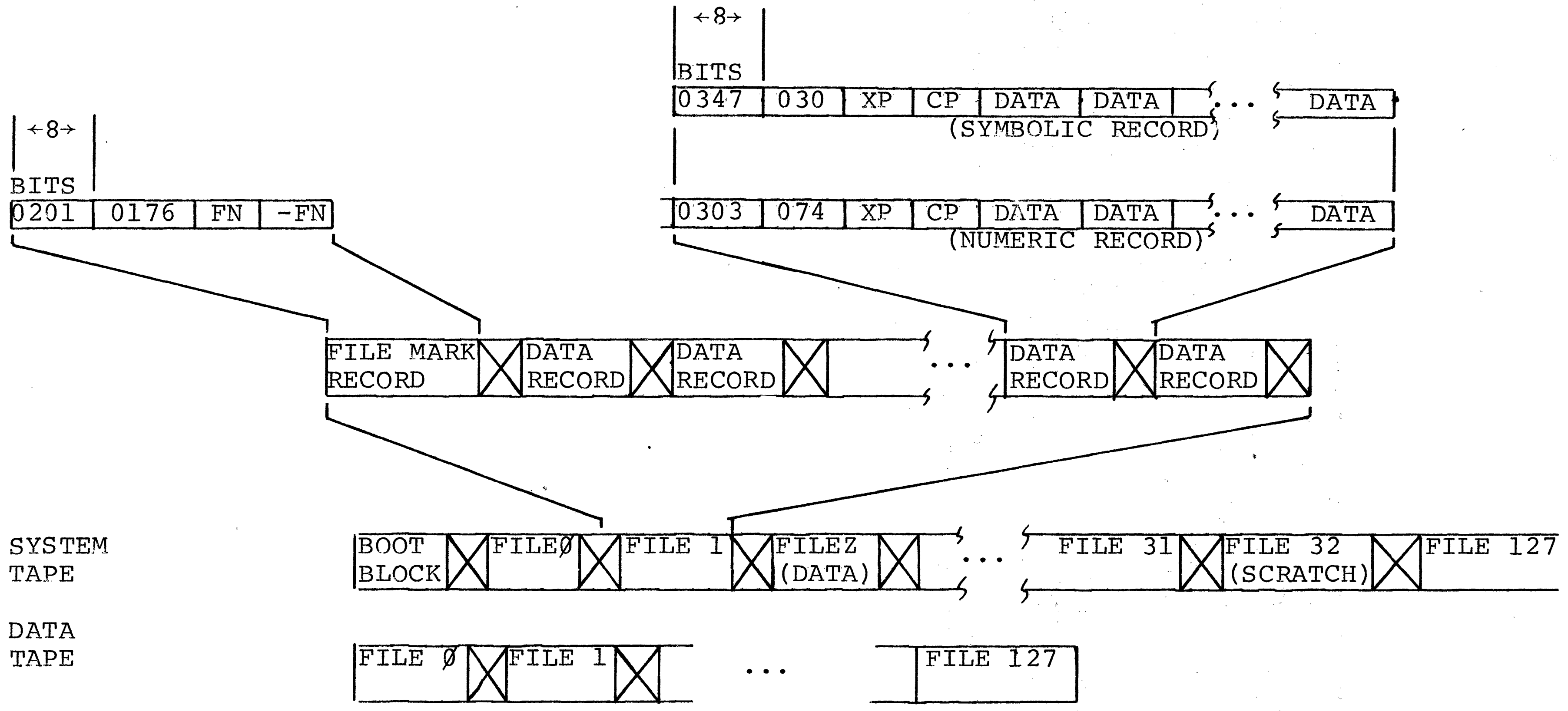
DATA TAPE: FILE0/FILE1/.../FILE127

2.4 LOADER ACTION

When the loader is told to load a given file, it begins searching the tape (the loader can load files from either deck, depending upon which entry point is used) forward until it finds a file marker record. Note that all records passed over must have a valid type number pair or an error recovery procedure will be initiated which will try up to three times to read the record correctly and then make an error exit if failure occured all three times. Upon finding a file marker, the loader determines, from the number in that marker, whether the tape is positioned to the correct place (the number is equal to that requested), is not positioned far enough forward (the number is less than that requested). If the tape is positioned to the correct place, the loader proceeds to load all of the numeric records it finds, obtaining the memory address of where it is to put the data from the beginning of each record, (symbolic records are ignored) until it runs across another file marker. At this point it stops the tape (which was in slew forward mode) and backs up over the file marker so a succeeding call on the loader would cause a file to be found immediately. If there were no numeric records in the file, an error return is made. If the tape is not positioned too far forward, the loader enters a reverse search mode. If, in this mode, the loader finds a file marker that indicates that the tape is not positioned backward far enough, the loader continues searching in the reverse mode for the next file. If, however, a file marker is found that indicates that the tape has been positioned too far backward, the loader decides that the file is not on the tape and makes an error return. Error returns are also made if a record cannot be read without a parity failure or type indicator discrepancy (the two characters are either not the one's complement of each other or are not one of the three special numbers) occurring in all three trials or if loading the record would overstore the loader. In all of these cases, the carry condition will be true (a satisfactory load always rendering the carry condition false) and the tape will be positioned after any offending record.

2.5 PARITY CHECKING

The third and fourth bytes of every data record contain longitudinal parity checks. These bytes are set up by the record generation program such that the following exclusive OR sums will yield zeros: the first byte with all the data characters (data characters start with the fifth byte of the record and proceed to the end) and the second byte with the same characters except the sum is shifted right circularly one place after each exclusive OR. In the case of symbolic records, the additional condition of the vertical parity of each character being odd must also be met. One thing not mentioned in the discussion of the loader was that the first four data characters (fifth through eighth bytes in the record) are not really data but are the MSP and LSP of the starting memory address followed by the one's complement of the MSP and LSP of the starting memory address of where the data is to be loaded.



☒	- INTER-RECORD GAP
FN	- FILE MARK NUMBER
XP	- "EXCLUSIVE OR" PARITY
CP	- CIRCULAR PARITY

<u>TAPE FORMAT</u>

CASSETTE TAPE OPERATING SYSTEM

SECTION III

OPERATING SYSTEM COMMANDS

3.0 KEYBOARD FACILITIES (OPERATING SYSTEM COMMAND HANDLER)

The operating system contains a program which will interpret user commands given at the keyboard and perform the tasks indicated. These tasks mainly involve copying new files from the data tape, deleting and updating files on the system tape, and executing programs kept in these files, as well as several other functions.

3.1 THE CATALOG

The operating system maintains a catalog of names and numbers which correspond to the files on the system tape. This catalog may be used in manipulating the files from the keyboard or in symbolically calling in overlays using the symbolic loader from a user program.

3.1.1 CATALOG CHARACTERISTICS

Each name in the catalog must start with a letter and may additionally contain from one to five alpha-numeric characters. There is room in the catalog for up to fourteen names so there is a limit of fourteen named files on one system tape. Another 16 files may be assigned as numeric files (020 - 037). Numeric files are specified by preceding the octal physical file number by a '#'. The symbolic loader contains routines which will look up a given name in the catalog and load the corresponding file. This same lookup routine is used by the command handler and is labled LOOKUP.

3.2 SYNTAX RULES AND ENTRY ERROR ACTIONS

The command input format is purposely made quite strict to reduce the chance of causing unwanted action which could be catastrophic to the user's data. The command must start with the first character entered (leading spaces are illegal) and any alphabetic after the third character is ignored (thus DEBACLE will be interpreted as the DEB command just as well as DEBUG). The first non-alphabetic character must be either an ENTER, a space or a dash (minus sign). Some commands will not allow the ENTER but typing a non-alphabetic other than these three will always net you an error message of WHAT? This will also appear if a command that has legal syntax but is not one of those defined is entered. If the command is to be parameterized, the first name or number must follow the dash or space immediately and must be terminated with an ENTER if that is the only parameter. Names must start with an alphabetic but may contain any number of alphanumeric characters even though all after the six will be ignored. File numbers must be preceeded immediately by a pound sign (#) and must be between 020 and 037 octal. If the command has more than one parameter, the last must be terminated by an ENTER. Parameters must be separated by commas with no embedded blanks. If a name is not supplied but the command requires one, the error message NAME REQUIRED will be displayed. If the name given is required to be in the file catalog but is not, the error message NO SUCH NAME will be displayed. If the inverse is true, the error message NAME IN USE will be displayed.

3.3 OPERATING SYSTEM COMMAND INSTRUCTIONS

The following paragraphs described the usage and effect of each command in the system. Each paragraph is titled by what must be entered to used the corresponding routine. Note for clarity, more than just the necessary three characters have been shown. For an abreviated description of commands see Appendix A.

3.3.1 CATALOG

The CATALOG command lists the names and numbers of all files that are currently on the system tape. They are listed across the screen in the physical order in which the files appear on the tape. Any parameters supplied are ignored.

3.3.2 NAME (old), (new)

The NAME command will change the name of the file specified by the first name given to the second name given. This command requires that the first and not the second name be in the symbolic catalog. The catalog file on the system tape (a one record file [number one] that immediately follows the operating system file) will be overwritten with the new catalog. Note that this leaves the system tape positioned before the file marker of any existing first cataloged file. This operation is performed by all commands that change the catalog.

3.3.3 RUN (name) or RUN #(number)

The RUN command uses the loader to load the file specified by the name or number given and then transfers processor control to the starting address indicated to the loader by the file information. Note that it is the responsibility of the loaded program to return to or reload the operating system if this is desired.

There is a special case to the RUN command that breaks the general syntax rules.

If the name consists of exactly one asterisk terminated by an ENTER (RUN *), the loader will be directed to load physical file 1 from the data tape. This provision is made to allow the user to run a program he has generated without having to load it onto the system tape. This, along with the F command in the debugging tool eliminates a loss of tape movement when debugging programs.

Symbolic files in the named catalog cannot be run.

3.3.4 IN (name), IN (number) or IN $

Note that exactly the characters shown must be typed to execute this command since the space which must be the third command character will also terminate the command. For a named file where there are no numbered files present, this command will position the system tape after the last cataloged file and the data tape to the beginning of physical file 1. (The data tape convention is that physical file 0 will be the first piece of information on the tape, containing the users symbolic data for a given program, and that physical file 1 will be the second piece of information on the tape and will contain the users object data for a given program, and all tape after this is to be considered a scratch area which is properly terminated by physical file 127 to indicate the logical end of the tape.) The command then copies all records in the file from the data tape onto the system tape creating a file on the system tape (a file marker being written before the data was copied) which has the next available physical file number. Following this new file, file marker 32 and 127 are written on the system tape to indicate the new start of system scratch and logical end of tape. If the system tape contained no cataloged files before this command was issued, the file entered will be physical file 2 and immediately follow the catalog file. After the new file has been written, the new name is entered into the catalog and the catalog file is updated as in the NAME command. Note that if the catalog was full when the command was entered, the error message LIBRARY FULL will be displayed and no other action will occur. The name supplied must not already be in the catalog. If a numbered file was specified and there are no numbered files greater than the one in the command, the action is as described above with the exception that the file marker written on the system tape will be the number specified in the command and not the next available number. No name will be entered but a bit will be set in the numeric catalog (two bytes in which every bit represents a possible numeric file, if the corresponding bit is a one, the file is present on the system). If a name is specified and there are files present with higher file numbers, this command proceeds like the INSERT command.

There is a special case of the IN command that breaks the general syntax rule.

If the name consists of exactly one dollar sign terminated by an ENTER (IN $), the front tape will be assumed to be a CTOS or a tape created by the OUT $ command. This command will position both the system tape and the data tape to file 1 (the catalog) and will copy the entire data tape up to file 32 onto the system tape. Following these new files, file markers 32 and 127 are written on the system tape. This command, therefore, replaces the catalog and all files on the original system. If the tape being IN'd is a CTOS tape, it must be positioned past the loader and file 0 marker since the file handling routines cannot process the loader.

3.3.5 OUT (name), OUT #(number), OUT $, OUT *, OUT ! or OUT OS$

The OUT command first executes the PREPARE command to provide itself with a null data tape which can be handled by the file handling routines. It then positions the system tape to the beginning of the given file (the name or number must have been in the catalog) and the data tape to the beginning of physical file one and copies all the records in the file on the system tape onto the data tape. It then places a file marker 127 on the data tape and quits. This command is provided to allow moving a file from one system tape to another through the associated use of the IN command. Outed non-symbolic files are written as file 1. Symbolic files are written as file 0 with a file 1 following.

There are four special cases to the OUT command that break the general syntax rules.

If the name consists of exactly one dollar sign terminated by an ENTER (OUT $) then an exact copy is made of the system tape up to file marker 32 at which time the copy is terminated by file markers 32 and 127 (which causes any scratch data on the old system tape to be removed).

If the name is exactly one asterisk terminated by an ENTER (OUT *) the action is similar to the previous case except physical file 0 and 1 (namely, the operating system and catalog) are deleted and the file numbers of all following named data files are lowered by two. Numeric files numbers are preserved unless there are no named files, in which case, the first numeric file is written as file 0. Note that if this tape is now bootstrap loaded, the first program loaded will be what was the first file cataloged in the operating system. This is most useful in preparing bootstrap tapes that will be used in machines with less than 8K of memory.

If the name is exactly OS$ followed by an ENTER (OUT OS$), a CTOS operating system tape with a blank catalog will be generated. The original catalog will be reread at the completion of this command.

If the name is exactly ! (OUT !), a three byte JMP DEBUG$ (016453) record will be written on the front deck followed by file 32 and 127 markers.

3.3.6 DELETE (name) or DELETE #(number)

The DELETE command takes two different courses of action depending on whether or not the file deleted is the last one cataloged. If it is, the system tape is moved to the end of the next to the last cataloged file and file markers 32 and 127 are written, thus logically destroying the last file. The name is then deleted from the catalog and the catalog file is updated. If the file is not the last one cataloged, the PREPARE command called to obtain a fresh data tape, as in the OUT command, and the system tape is positioned to the end of the named file. The rest of the system tape (up to file 32 marker) is then copied onto the data tape and the data tape is terminated with a file marker 127. Note that the data tape file numbers start out at one and increase by one for each succeeding file copied onto the data tape. Numeric file numbers (020 - 037) are maintained. These numbers are not used for named files since all the copy back part needs to know is file delimitation since it is getting its file number information from catalog positions. The copying onto data tape is followed by the system tape being positioned to the end of the-file before the one named and the data tape being positioned to the beginning of file one. A file marker having a value one greater than the previous marker is then written on the system tape with every file marker encountered on the data tape causing a file marker of value one greater then the previous marker to be written on the system tape. This process terminates when a file marker 127 is encountered on the data tape which causes file markers 32 and 127 to be written on the system tape. The given name is deleted from the catalog, all following entries are dropped down one place to correspond to the similar shift in file numbers that took place, and the catalog file is updated.

3.3.7 REPLACE (name) or REPLACE #(number)

The REPLACE command is quite similar to the DELETE command except that instead of preparing the data tape with the

PREPARE command, it positions it to the end of file 1 and then writes a file marker 2. Now, copying all the files after the named one onto the data tape in a fashion as in the DELETE command will replace the named file by file 1 on the data tape, with any necessary physical expansion or contraction taking place. Even though the catalog is not changed in this operation, it is updated anyway since this is an easy way to position the system tape to a place before file marker 127. Without this, a succeeding call on the loader would run into trouble since the system tape would be off the logical end of the tape. The loader starts with a forward search because the very first time it is used, the tape is positioned just after the boot block and a backward search for a file marker would cause trouble. The operating system routine which searches for files can start with a reverse search to avoid the problem since the tape will never be resting before file zero.

3.3.8 AUTO or AUTO (name) or AUTO #(number)

There is a word in the catalog which contains the physical file number of a file which should be loaded and executed immediately upon loading and execution of the operating system. This enables a user program to be run after restart without interaction with the operating system being required. If this word is a zero or the keyboard switch is being depressed upon initial execution of the operating system, the normal entry is made in the operating system and the start up message and response request are displayed.

If the AUTO command is given with no name and the auto pointer is zero than the error message NAME REQUIRED will be displayed. Otherwise, the name of the file being pointed to will be displayed in the message AUTO SET TO (name). If the auto command is given with a name (which must be in the catalog) and the auto pointer is a zero, the pointer will be changed to the corresponding file number and the catalog (which contains the pointer) will be updated. If the auto pointer is non-zero, the name is ignored and the AUTO SET TO (name) will be displayed as in the no-name case.

3.3.9 MANUAL

The MANUAL command will zero the auto pointer and update the catalog if the auto pointer was non-zero. Otherwise, the message AUTO NOT SET will be displayed.

3.3.10 PREPARE

The PREPARE command first asks the operator if the data tape contains anything of value and then halts. (Note that the auto-restart tab should not be broken out of the system tape because it will prevent use of the OUT, DELETE, or PREPARE commands since halting the processor will cause an auto-restart.) After the operator hits the RUN button as a response, it is assumed that the data tape is of no value as it is rewound and file markers 0, 1 and 127 are written on it. This is needed since the operating system routines require file markers for which they can search in using the data tape.

3.3.11 REWIND

This command rewinds the data tape on the front deck.

3.3.12 CHOP (name) or CHOP #(number)

The CHOP command deletes the specified file and all subsequent files.

3.3.13 INSERT (new,old)

The INSERT command places the object file from the data tape onto the system tape before the "old" file named in the command. This is accomplished by copying the files from the system tape starting at the 'old' file through file 32 onto the data tape at the end of file 1; and then copying back the files starting at file 1, the "new" named file. The catalog is rewritten at the end of the operation.

3.3.14 APPEND (name) or APPEND #(number)

The APPEND command appends the object file from the data tape onto the end of the named file on the CTOS system tape. Like the DELETE command, it has two possible courses of action, depending on whether or not the file being appended is the last cataloged file. If it is, the tape is positioned to the end of the cataloged file and a new object file is copied from the front deck. If the named file is not the last cataloged file, the operation proceeds like the REPLACE except that the system tape is positioned to the end of the named file before the update is performed.

3.3.15 LGO (name [,name, name...])

The LGO command makes a tape with a loader and the specified (named or numbered) file (s) in the sequence named in the command. The files will have sequential file markers starting with 0. There is a limitation of 55 characters on the command length, thus to name many files in the LGO command it may be necessary to temporarily rename the files with one character label. LGO-* is <u>not</u> permitted. OUT-* has the desired effect of generating a load and go tape of all cataloged files (note, OUT-* preserves numeric file numbers 020 - 037). Files 32 and 127 are written at the end.

3.3.16 SYMBOLIC (name) or SYMBOLIC #(number)

The SYMBOLIC command adds a compressed source file (file 0) to the CTOS tape in a fashion similar to IN. The name in the internal catalog will have an 'S' in the seventh (not displayed) position to identify the file as symbolic. If a file number is specified, no indicator will be set to distinguish symbolic from non-symbolic files.

3.3.17 SREPLACE (name) or SREPLACE #(number)

The SREPLACE, symbolic replace, command is performed exactly as the REPLACE except the compressed source file (file 0) is used instead of the object file. File 1 will be overwritten if the file being replaced is not the last one on the system tape.

3.3.18 SINSERT (new), (old)

The SINSERT, symbolic insert, command is performed exactly as the INSERT except the compressed source file is inserted instead of the object file. File 1 will be overwritten.

3.3.19 ATTACH (name [, name, name...])

The ATTACH command positions the front tape to the end of file 0 and (without file markers) copies specified file(s) from the system tape to the front deck. Numeric files may be specified. Files are not checked to insure symbolic format. This is useful for combining subroutine source files with program source.

3.3.20 ATOBJECT (name [, name, name...])

The ATO command positions the data tape to the end of file 1 and proceeds as the ATTACH command.

3.3.21 FADVANCE

Advances the front cassette past the next file mark.

3.3.22 RADVANCE

Advances the front tape past one record - the record format is ignored; therefore, FAD or RAD may be used to position a rewound CTOS tape for use with the IN-$ command.

3.3.23 BACKUP

Backspaces the front tape one record.

3.3.24 DEBUG

The debugging program allows the user to observe and modify any location in memory to load files from either the system or data tapes, and to start execution at any place in memory. This allows him to load and debug programs with surprising ease. The major debugging technique is to insert RETURN instructions in critical places in memory so one routine at a time may be checked using the CALL command. All but two (user specifiable) of the registers may be saved upon return from the program being tested, allowing the user to determine if the proper actions are taking place by observing critical register and memory values. The registers A, B, C, D, E (subject to the H and L commands) are stored in locations 016770, 016771, 016772, 016773 and 016774 respectively upon a return to Debug from a program which was called by Debug.

3.3.24.1 INPUT SYNTAX AND ERROR ACTION

The debugging program is entered from the command handler as explained in a later section or by processor execution control being passed to the location labeled DEBUG$. At this time the bottom line of the display will be erased and the current location and its contents will be displayed there. The program is now ready to accept input in the format <number> <command>. Only sixteen bits of significance are kept for the input value. If more are entered, the first digits entered are lost. Some commands use only the lower order eight bits. The number is terminated by the first character that is not between zero and seven and this character is taken to be the command. Note that leading spaces are not permitted. This line is read in using the KEYIN$ routine discussed (in section 4) thus enabling the use of the backspace and cancel keys but requiring the ENTER key to be struck to obtain a response. In one case the ENTER character is the command and in some others the number is disregarded. If the command is not recognized, the program simply ignores it and the old current address and its contents are displayed again. After every command, control is returned to the entry point of the debugging program which will display the now current address and its contents.

3.3.24.2 THE CURRENT ADDRESS

Two memory locations in the debug routine contain an address (initialized to zero upon loading) which points to a memory location which is the current center of interest. Available commands allow one to change the contents of this memory location and move the pointer as well as perform other functions.

3.3.24.3 DEBUG COMMAND MEANINGS

The following is a list of each command character with its effect and the number (in parenthesis) of bits of the given number used:

F	load file one from the data tape
G	load from the system tape the file whose number is given (8)

ENTER	set the current address to the number given (16)
I	increment the current address by one (0)
D	decrement the current address by one (0)
M	change the current address contents to the number given (8)
.	do the M followed by the I command (8)
L	upon return from a C command, cause the L register to be stored into the register whose number is given (3)
H	same as the L command but for the H register (3)
C	execute a CALL instruction to the location whose number is given (16)
O	return to the operating system command handler (be sure it is there) (0)

3.3.25 LIST

The LIST command displays any tape. The tape must be placed in the front deck and previously positioned. If records are numeric (record type 0303), parity errors will be indicated by the 'Internal Error D' message - no reread will be performed. The first four bytes (0303/074/XP/CP) will not be displayed for numeric records.

The operator must answer the question 'Alpha or Octal?' by typing 'A' or 'O'. Any type of symoblic tape may be listed in the alpha mode, including compressed source tapes generated by the editor; however, compressed source blocks will not be linked together - each block will be displayed as an independent entity.

To initiate and continue the display type 'L'.

To advance to the next record, type 'N'.

To backspace a record, type 'B'.

To return to the command handler, type 'O'.

A maximum of 256 bytes can be displayed.

3.3.26 FIX

The FIX command permits the operator to look at and change an object file. The tape must be placed in the front deck and the operator must answer the question 'CTOS or OBJ?'. If 'C' is typed, the actual octal file number, 'FILE NO?' must be supplied; otherwise, the tape will be positioned to the object file (file 1). New blocks cannot be generated on a CTOS tape.

The routine is now ready to accept input in the format <NUMBER><COMMAND>. The number is assumed to be octal and the absence of any digits between zero and seven implies a value of zero for the number. Only sixteen bits of significance are kept for the input value. If more are entered, the first digits entered are lost. Some commands use only the lower order eight bits. The number is terminated by the first character that is not between zero and seven and this character is taken to be the command.

In one case the ENTER character is the command and in some others the number is disregarded. If the command is not recognized, the program simply ignores it and the old current address and its contents are displayed again.

Operation is similar to DEBUG. The following is a list of command characters with its effect and the number (in parenthesis) of bits of the given number used:

ENTER:

a) if no block is currently in memory, read object file until block containing given location is found, then display the contents of that location.
b) if a block is in memory and the location given is within the limits of the block, the contents of the location will be displayed.
c) if a block is in memory and the location given is not within the block limits, the current address will be set to the minimum or maximum address, its contents will be displayed and

a beep will sound.
d) if a new block is being created in memory and the given location is between the starting address and the current address + 1 both the current and maximum address will be set to the given location and its contents will be displayed. (16)

A	abort processing of current block. Reposition tape to beginning of file (Ø).
T	write block currently in memory out to tape over original block (Ø).
N	develop a new block with starting address given. The last block (which transfers control to the starting address) will be rewritten after the new block. The length of the new block is determined by the last address displayed before 'T' is typed (16). (Object tape only)
I	increment up to maximum address in current block. If new block, increase length of block (Ø).
D	decrement to minimum address of current block. If new block, decrement length of block (Ø).
M	change contents of displayed address to number given (8).
.	change contents of displayed address to number given and display contents of next location (See 'I' command) (8).
O	transfer control back to operating system command handler (Ø).
R	reset file number - aborts processing and re-asks the first question.

The tape will not be repositioned to the beginning when you type 'T' - so if you enter corrections in the same sequence they are arranged on tape, operation will be most efficient. If you need to access a previous block, type 'A' and then the address.

CASSETTE TAPE OPERATING SYSTEM

SECTION IV

SYSTEM SUBROUTINES

4.0 SYSTEM SUBROUTINES

Many of the subroutines resident in the operating system can be useful to user programs. Functions such as keyboard input, screen output, tape handling and linking can be performed using proper input parameters to these memory resident routines.

4.1 SYMBOLIC LOADER ROUTINES

4.1.1 UTILITY ROUTINES

Several utility routines contained in the symbolic loader area are a block transfer, labeled BLKTFR and a routine, labeled INCSWP, which increments the H and L register pair and then swaps it with the D and E register pair. The block transfer will move the number of characters specified by the entry value in the C register from a memory address starting with the entry values in the H and L registers to a memory address starting with the entry values in the D and E registers.

4.1.2 LOADING ROUTINES

To use the symbolic loading routine, one loads into the D and E registers the address of the six characters of the desired name (trailing blanks must be included) and calls MLOAD$. If the zero condition is false upon return, then the given name was not in the catalog. If the zero condition is true but the carry condition is false upon return, the laoder could not either find or correctly load the file requested. Note that one must be certain to place the call to MLOAD$ in a place that will not be overlayed since execution will resume following the CALL instruction after the file has been loaded.

4.1.3 OTHER SYMBOLIC LOADER FACILITIES

Another facility in the symbolic loader area will load and execute a file whose number (not name) is in the B register upon call. Calling MAUTO$ will load the file from the system tape and calling MAUT2$ will load the file from the data tape. If the loader could not either find or load the file, the operating system is automatically reloaded.

4.2 LOADER ROUTINES

There are several routines available in the loader that can be used without the symbolic loader interface. BOOT$ reloads the operating system from the rear deck. LOAD$ loads and executes the file specified in the B register from the rear deck. LOAD2$ loads and executes the file specified in the B register from the front deck.

4.3 OS UTILITY ROUTINES

INCHL increments the H and L registers by one. INDEX increments the H and L register by the value in the A register. DECHL decrements the H and L registers by one. SAVHL and RESHL are two routines to save and restore the H and L registers as long as no file handling is performed between the call to SAVHL and the call to RESHL.

4.4 KEYBOARD AND DISPLAY ROUTINES

The operating system contains facilities to ease the burden of communicating with the operator. Two routines exist. The first accepts the characters from the keyboard, displays them on the screen, and stores them in a memory buffer. The second writes a string of characters from a memory buffer onto the screen.

4.4.1 KEYBOARD INPUT

The keyboard input routine, labeled KEYIN$, accepts a specified maximum number of characters, given by the entry value of the C register, from the keyboard and puts them into memory starting at the entry value of the H and L registers and onto the screen at starting horizontal cursor position of the entry value of the D register and vertical cursor position (which cannot be changed during the course of one input) of the entry value of the E register. Note that if the cursor collides with the right edge of the screen during entry, characters other than backspace, cancel, and ENTER will not be accepted, although they will print over each other in the last display position. The ENTER character (015) terminates input and is stored in the memory buffer to specify the end of data but is not written to the screen. Hitting the backspace key will delete the last character entered and move the cursor appropriately while hitting the cancel key will delete all characters entered and also move the cursor appropriately. These two keys also back up the buffer memory pointer appropriately. Note that if one has typed a character at either the screen limit or at the maximum character count limit, hitting a backspace will cause the previous character to be erased and leave the last character still on the screen, although it will either not appear on the memory buffer or be after the 015.

4.4.2 DISPLAY OUTPUT

The display routine, labeled DSPLY$, will display the string of characters stored in memory starting at the address which is the entry value of the H and L registers and terminating with a character whose numerical value is either a 003 (ETX) or 015 (ENTER). The cursor starts at the entry values in the D (horizontal) and E (vertical registers a cursor position that is off the screen will not be sent to the CRT) and stops after the last character printed if the terminating character was an 015. Note that, as in KEYIN the cursor stops at the right edge of the screen and the characters overwrite each other if more are available after collision. Also note that if display was occuring on the bottom line and the terminating character is an 015, then the whole screen is rolled up to force the existance of a following line and the information that was at the top of the screen is lost. After return from the display routine, the H and L registers will point to the location after the terminating character and the D and E registers will reflect the current cursor position. The cursor will be off while the display routine is writing, but it is turned back on upon exit even if it was off upon entry. Other special control characters can cause cursor positioning line/frame erasure, and screen roll-up:

011	a new horizontal position (0 to 79) follows
013	a new vertical position (0 to 11) follows
021	erase to the end of the frame
022	erase to the end of the line
023	roll the screen up one line

4.5 OPERATING SYSTEM FILE HANDLER

The operating system contains a set of routines which will perform all of the various input/output functions needed to maintain the files of data on the tapes. These routines are packed in the upper 2K of memory and are made available to the user if he wishes to handle his mass storage problems in conformance with the conventions of the operating system. All routines are uniformly parameterized and are accessed through an entry point table (a group of JUMP instructions to the actual routine locations) so any update to the operating system will not have any effect upon the user's code.

4.5.1 ROUTINE PARAMETERIZATION

Routine parameterization consists of a memory location in the D (MSB) and E (LSB) registers of the first byte of a group of four bytes (called a packet) which parameterizes the call more explicitly. This method reduces the number of memory locations required to perform a routine call since, in a typical program, one needs only a few different packets but will have many different calls. The parameterization of some routines is not as extensive as that of others, but the same packet can generally be used for the different calls when they are affecting the same file.

4.5.2 LOGICAL FILE NUMBERS

The first byte in the packet is the logical file number and must be between zero and seven or an internal error H will occur upon calling the routine. The second and third bytes in the packet contain the LSB and MSB (respectively) of the first location in memory to be used as a data buffer. Actually, the two bytes previous to this location will be used by some of the routines as discussed later. This data buffer may be located anywhere in memory. The fourth byte in the packet specifies the length of the data buffer when numeric data is being handled. Note that using only one byte for the length implies that numeric records may not contain more than 256 data bytes. Actually the maximum number of data bytes specified may not be greater than 254 for reasons that are made clear in the numeric routine instructions. The four bytes of the packet may be located anywhere in memory.

4.5.3 PHYSICAL DEVICE AND FILE NUMBERS

The logical file number specified in the packet is converted by each routine, via an internal transformation table, into physical file and device numbers. The physical device number specifies whether the operation is to be performed on deck one (rear) or deck two (front) and the physical file number specifies which file is to be treated on the given deck. Actually, not all routines use all of this information since, for instance, when one is reading records from a file he assumes that he is using the file to which the tape was last positioned. The internal transformation table is initialized at load time to the following values:

LOGICAL FILE	PHYSICAL FILE	PHYSICAL DEVICE	GENERAL USE
0	0	0	Unassigned
1	0	1	General deck one
2	0	2	General deck two
3	1	1	CTOS catalog

LOGICAL FILE	PHYSICAL FILE	PHYSICAL DEVICE	GENERAL USE
4	0	2	Symbolic data
5	1	2	Object data
6	0	0	Unassigned
7	32	1	System scratch

It is shown that logical files 1 and 2 are specified for use of any physical file, even though 0 is shown in the table. This can be done by use of a routine that will change the physical file number of a given logical file. A routine also exists to allow the physical device number to be changed, thus allowing the user to set up logical files in any physical configuration needed. Note, however, that one must have logical files 1 through 5 and 7 in the state shown (except for the physical device numbers of logical files 1 and 2) if one returns control to the operating system command handler, since the loaded values are assumed by this program. Logical files 0 and 6 may be used freely but must be set before the first call utilizing them. The following is an example of a packet usage as it would be expressed in the assembler: (Note all calls to CTOS tape routines must as in the following example, be preceeded by a DE to the first byte of the packet. Note also that the packet consists of 4 bytes: Logical file number, LSP of buffer, MSP of buffer and length of buffer)

```
                LA      2          Set up logical file six
                DE      PACKET     to be used as physical
                CALL    CPDN$      file 3 on the front deck
                LA      3
                DE      PACKET
                CALL    CPFN$
                DE      PACKET     Position to the beginning
                CALL    PBOF$      of the file
LOOP            DE      PACKET     Read a record of symbolic
                CALL    SSFR$      Into BUFFER
                JTC     DONE       Quit it to the next file
                                   marker
                JFZ     TERR       Exit if type error
                .
                .                  Action taken for each record
                .
```

```
          JMP     LOOP
DONE      .
          .                 Action taken when file
          .                 completely in
TERR      .
          .                 Type error action
          .
          DC      0,0       Room for parity check
                            generation
BUFFER    SKIP    128       Buffer area
PACKET    DC      6         Logical file 6
          DA      BUFFER    Buffer address
          DC      0         Length (not used)
```

4.5.4 ROUTINE USAGE INSTRUCTIONS

To use a routine, one sets up whatever is required for proper parameterization and then calls the desired location in the entry point table. The locations are labeled with the first word in the following paragraph titles followed by a $. For example, to call the serial numeric file read, one would say CALL SNFR$. The routine will either perform the requested task or take one of two error exit paths. The first path is taken in the case of fatal errors, for which it is decided that the only recourse is to reload the operating system. This is called an internal error and the message INTERNAL ERROR (letter) is written on the bottom line of the display before the system is reloaded. The various letters which may appear are the following:

A	Illegal device specification
B	Illegal record format
D	Unrecoverable parity error
G	Unfindable file
H	Illegal logical file specification

The other path is non-fatal and simply returns with certain condition flags in states other than normal to indicate that something unusual happened. Since every routine uses a common subroutine (labeled GETPKT) to get the parameters from the specified packet, common internal errors can occur. If the logical file number is not between zero and seven, an internal error H occurs. If the physical device number is not either a 1 or a 2, an internal error A occurs. Other than for these error actions, the following paragraphs described the effects of and the exact parameterization needed for each routine.

4.5.4.1 SNFR - SERIAL NUMERIC FILE READ

This routine reads the next record from the specified device. If the record is of symbolic type, the zero and carry condition are set false and return occurs with no parity checking or data storage being performed. If the record is a file marker, the carry condition is set true and the tape is backed up to where it was before the routine was called. Again, return occurs with no data storage being performed. If the type is numeric, the two parity bytes followed by the data are read into the buffer. If the parity checking fails or the record type is bad, three efforts are made at reading the record by backing up to its beginning and starting over. If recovery is not made in one of these efforts, an internal error D occurs. If the record is read successfully, return occurs with the zero condition true, the carry condition false, and the H and L registers containing the memory location of the byte following the last one loaded from the tape. To calculate the length of the buffer area used, one must subtract the buffer starting address from returned values in the H and L registers. Remember that the first two characters in the buffer are not data characters but are the two longitudinal check sums. To obtain the number of data characters loaded, one must subtract the buffer starting address plus two from the returned values in the H and L registers. The parity checks are stored because the SBFW routine uses them instead of regenerating them from the data, thus shortening the time required to copy numeric records from one deck to the other.

4.5.4.2 SSFR - SERIAL SYMBOLIC FILE READ

This routine reads the next record from the specified device. If the record is of type numeric or file marker, the action taken will be the same as when SNFR reads a symbolic or file marker record. Action similar to that taken by SNFR is also taken if parity or type faults occur. If the record is read satisfactorily, only data characters will be in the buffer starting at the address specified. An 015 will mark the end of the data string and all vertical parity bits will be zero. The same normal exit conditions as in SNFR will occur.

4.5.4.3 SBFW - SERIAL BLOCK FILE WRITE

This routine writes a record of type numeric on the specified device. The total number of bytes, including the parity initialization sums as the first two, must be in the fourth byte of the packet. Note that inclusion of the parity initialization sums implies that the total number of actual data bytes cannot exceed 254. This routine assumes that the first two bytes in the buffer are the correct parity initialization sums since it does not generate them from the data. There are no error exits from this routine which implies that writing off the end of the tape will not be caught and that-read after-write checking is not performed.

4.5.4.4 SNFW - SERIAL NUMERIC FILE WRITE

This routine performs in a fashion similar to the SBFW routine except the two parity bytes are not included in the data buffer and the length specifies the number of actual data characters. The routine generates the two longitudinal parity sum initialization values and inserts them in the two locations preceeding the buffer. It then writes on the specified device a record of type numeric containing the two parity bytes generated, followed by the number of data bytes specified. Note that the length is adjusted to accomodate the two parity bytes so, as in the SBFW routine, only 254 actual data bytes may be written. If one specifies a length of 255 or 0, the only bytes (besides the record type) written on the tape will be respectively the first or both parity initialization sums. No error exits are made from this routine.

4.5.4.5 SSFW - SERIAL SYMBOLIC FILE WRITE

This routine performs in a fashion similar to the SNFW routine except that an 015 character in the data string rather than a specified value is used to determine the buffer length, vertical (in addition to longitudinal) parity generation is performed, and a record of type symbolic rather than numeric is written. The terminating 015 character is not included in the set of characters written to the tape, but remember that it will appear again if the SSFR routine is used to read the record.

4.5.4.6 PEOF - POSITION TO END OF FILE

This routine searches forward on the specified device until it finds a file marker. It then backspaces the tape until it is between the next to the last and the last record in the file. It then forward spaces the tape one record which puts it at the end of the file, having arrived there via forward tape motion. This forward arrival is important to observe when one plans to append one record after another and still maintain physical interrecord gap integrity. Note that every record passed over by the PEOF routine must have a valid record type or it will be read again in action similar to parity failure action in the SNFR routine.

4.5.4.7 PBOF - POSITION TO BEGINNING OF FILE

This routine searches for a file marker in a fashion similar to the loader except it starts by searching backwards. The file number searched for is specified by the physical file number supplied by the generalized parameterization. Note that since this routine starts by searching backwards, it will not decide that the requested file is not on the tape until it has found in the search forward mode a file marker that specifies a file number greater than the one desired, if indeed the file is not on the tape. Also note that if the leader is found in the search backward mode, the tape is positioned forward past the first record and the backward search is continued. If the first record is not a file marker (operating system convention requires it to be) or is a file marker whose value is greater than the one desired, the first record on the tape will be passed over back and forth until external intervention is imposed. Otherwise, all search rules and error exit conditions of the loader routine apply here. If, upon return, the carry condition is true, then the file was not found. Otherwise, the tape will be positioned at the interrecord gap following the file marker, having approached that point with forward tape motion for the reasons expressed in the PEOF routine.

4.5.4.8 BSP - BACKSPACE

This routine simply backspaces the tape one record using the hardware backspace function. No checking is made to see if the record was of proper type or if the tape ran onto the leader.

4.5.4.9 CPDN - CHANGE PHYSICAL DEVICE NUMBER

This routine stores the entry value of the A register (note the break from generalized parameterization) into the physical device number entry for the specified logical file in the internal transformation table.

4.5.4.10 CPFN - CHANGE PHYSCIAL FILE NUMBER

This routine stores the entry value of the A register (note the break from generalized parameterization) into the physical file number entry for the specified logical file in the internal transformation table.

4.5.4.11 TRW - TAPE REWIND

This routine performs a hardware high speed rewind of only the front deck. If the rear deck (physical device 1) is specified, an internal error A will occur. Upon exit from this routine, the tape will be positioned to the clear leader.

4.5.4.12 TFNR - TAPE FILE NUMBER READ

This routine acts in a fashion similar to PEOF until it finds the file marker. At this point, it simply reads the value of that marker and leaves the tape positioned after the marker record. The value read is return to the C register. Error exits similar to the PEOF routine can occur.

4.5.4.13 TFNW - TAPE FILE NUMBER WRITE

This routine will write on the specified deck the special four byte file marker record containing the physical file number specified. No error exits will occur.

APPENDIX A

SYSTEM SUBROUTINE LOCATIONS

SNFR$	EQU	014000	Serial Numeric File Read
SSFR$	EQU	014003	Serial Symbolic File Read
SBFW$	EQU	014006	Serial Block File Write
SNFW$	EQU	014011	Serial Numeric File Write
SSFW$	EQU	014014	Serial Symbolic File Write
PEOF$	EQU	014017	Position to End-Of-File
PBOF$	EQU	014022	Position to Beginning-of-File
BSP$	EQU	014025	Backspace One Record
CPDN$	EQU	014030	Change Physical Device Number
CPFN$	EQU	014033	Change Physical File Number
TRW$	EQU	014036	Tape Rewind
TFNR$	EQU	014041	Tape File Number Read
TFNW$	EQU	014044	Tape File Number Write
KEYIN$	EQU	017000	Keyboard Entry Routine
DSPLY$	EQU	017151	CRT Display Routine
INCHL	EQU	017353	Increment H & L
DECHL	EQU	017364	Decrement H & L
BLKTFR	EQU	017745	Transfer Block from HL to DE C Characters

SYSTEM SUBROUTINE LOCATIONS

INCSWP	EQU	017765	Increment HL & Swap it with DE
BOOT$	EQU	064	Reload OS
MLOAD$	EQU	017620	Symbolic File Loader
MAUTO$	EQU	017601	Load and Execute - OS tape
MAUT2$	EQU	017612	Load and Execute - Deck 2
LOAD$	EQU	0100	Load and Execute - Deck 2
LOAD2$	EQU	0112	Load and Execute - Deck 2

APPENDIX B

OPERATING SYSTEM COMMANDS

COMMAND	FUNCTION
CAT	Lists names and numbers of all files on the system tape.
NAME (old),(new)	Changes the name of the file specified in the catalog.
RUN (name) or RUN #(number)	Loads and executes specified file
RUN *	Loads and executes file 1 on the front deck
IN (name)	Adds file 1 from the front deck to the system tape after the last cataloged file.
IN #(number)	Adds file 1 from the front deck to the system tape in the specified file position.
IN $	Copies the catalog and all program files from the front deck to the rear deck.
OUT (name) or OUT #(number)	Moves specified file from the system tape to the front deck. File 0 if symbolic, file 1 if non-symbolic.
OUT $	Copies rear deck to front deck (in total).
OUT *	Copies rear deck to front deck (excluding catalog and operating system). Makes a load and go tape.
OUT OS$	Generates system tape with blank catalog on front deck.

OPERATING SYSTEM COMMANDS

COMMAND	FUNCTION
OUT !	Generates JMP DEBUG$ tape on front deck.
DELETE (name) or DELETE #(number)	Removes specified file from the system tape.
REPLACE (name) or REPLACE #(number)	Replaces specified file with file 1 from the front deck.
AUTO	Displays current file in AUTO status.
AUTO (name) or AUTO #(number)	Changes catalog to reflect that the program or file specified will automatically be loaded and executed upon restart.
MANUAL	Permanently removes the affect of a previous AUTO command.
PREPARE	Rewinds front deck and writes a null data tape.
REWIND	Rewinds front deck.
CHOP (name) or CHOP #(number)	Deletes specified file and all subsequent files.
INSERT (new),(old)	Inserts file 1 from front deck in front of the specified file.
APPEND (name) or APPEND #(number)	Adds the object file 1 from the front deck to the end of the existing file specified.
LGO (name,name...)	Generates a load and go tape with the programs in the order specified.
SYMBOLIC (name) or SYMBOLIC #(number)	Adds file 0 from the front deck to the system tape after the last cataloged file.

OPERATING SYSTEM COMMANDS

COMMAND	FUNCTION
SREPLACE (name) or SREPLACE #(number)	Replaces the symbolic file specified with file 0 from the front deck.
SINSERT (new),(old)	Inserts file 0 from the front deck to the specified position on the rear deck.
ATTACH (name,name...)	Adds specified symbolic files from the system tape to the end of the current file 0 on the front deck.
ATOBJECT (name,name...)	Adds specified non-symbolic file from the system tape to the end of the current file 1 on the front deck.
FADVANCE	Advances the front cassette past the next file mark.
RADVANCE	Advances the front cassette past the next record.
BACKUP	Backspaces the front cassette one record.
DEBUG	Enters DEBUG routine in the operating system.
LIST	Enters LIST routine which displays any tape on the CRT.
FIX	Enters FIX routine which permits permanent modification of program tapes.

APPENDIX C

The following chart indicates differences and limitations of operating systems released prior to CTOS 3.1.

CTOS VERSION/ REVISION	LOADER	CTOS STARTING LOCATION	CATALOG	COMMAND NOT AVAILABLE
CTOS 1.2	8K limit	05000	No numeric File space	No numeric file handling commands OUT OS$ OUT ! REWIND FADVANCE RADVANCE LIST FIX ATOBJECT BACKUP IN $ CHOP * LGO * INSERT * SYMBOLIC * SREPLACE * ATTACH *
CTOS 2.2	8K limit	03400		LIST FIX OUT !
CTOS 2.3		03400		LIST FIX OUT !

*Available if overlay SOSX used.

Datapoint

Datapoint Corporation
9725 Datapoint Drive
San Antonio, Texas 78284
512/696-4520