GO:CMD^{TM}

Golden Oldies:
Command Utility Package
OPERATOR MANUAL

GO:CMD^{TM}

Golden Oldies:
Command Utility Package
OPERATOR MANUAL

Revision 1.0.0 11/17/88

MISOSYS, Inc
PO Box 239
Sterling, VA 22170-0239
703450-4181

Golden Oldies: Command Utility Package

SOFTWARE LICENSE AGREEMENT

MISOSYS, Inc., authorizes you to use this software on only one computer at a time. You are authorized to make archived copies of the software for the sole purpose of backing up your software.

MISOSYS, Inc., warrants the physical diskette and physical documentation to be free of defects in materials and workmanship for a period of 30 days from the date of purchase. Upon notification of defects in material or workmanship within the warranty period, MISOSYS, Inc., will replace the defective documentation or diskette.

> **MISOSYS, Inc disclaims all other warranties, expressed or implied, including but not limited to any implied warranty of merchantability and/or fitness for particular purpose. Under no circumstances shall MISOSYS, Inc be liable for any loss of profit or any other damage, including but not limited to special, incidental, exemplary, consequential or other damages.**

Golden Oldies: Command Utility Package

Table of Contents

Golden Oldies: Command Utility Package

General Description

COMP: This program compares two files, parts of files, diskettes, or parts of diskettes for a character for character match.

FASTBACK: is a utility program that will copy a file larger than can be held on a single floppy disk to multiple diskettes.

FASTREAD: will restore a set of floppy diskettes back to a file on a hard drive.

FED2: Allows the user to access a disk file to display, disassemble, and edit that file. It is a screen-oriented File EDitor to be used with a LS-DOS compatible operating system. Its wide range of capabilities make it an excellent tool for the advanced user. Its simplicity makes it easy to use for the novice.

IFC: IFC gives you the ability to perform file maintenance of moving files among disks while purging unneeded files. A menu-controlled screen provides the tools to easily list, copy, delete, or rename a file or groups of files. You will find IFC essential to the task of file maintenance.

PRO-CESS: A powerful maintenance tool for "CMD" or "CIM" type load module files. It provides for file appending, mapping, sorting, packing, offsetting, library member and partitioned data set member extraction, as well as the specified deletion of any load module record. PRO-CESS can convert "CMD" files to "CIM" files which are pure binary core-image constructs. It also provides the capability of converting "CIM" files to "CMD" files.

ZCAT: A very fast machine language program that creates and maintains a catalog of all files which reside on your LS-DOS 6.3 (or TRSDOS 6.x) formatted diskettes.

COMP

This program compares two files, parts of files, diskettes, or parts of diskettes for a character for character match. The proper syntax is:

```
COMP filespec1 TO filespec2 (parm,parm,...)
COMP :drive1 TO :drive2 (parm,parm,...)
```

The allowable parameters are:

Rec=	Starting record number of the filespecs at which the compare will begin default is 0
Num=	Number of records of a filespec or sectors of a disk to compare
All	Display each non-matching byte
Print	Send display to *PR (printer) as well as *DO (video screen)
Cyl=	Cylinder at which to start compare between two drives (default is 0)
Sec=	Starting sector of a diskette to compare (default is 0)

Abbr:	Rec=R, Num=N, All=A,
	Print=P, Cyl=C, Sec=S

This utility compares two files or two entire diskettes to determine whether or not the information in or on them is identical. It is usually performed after a BACKUP or a COPY to determine the validity of the data.

If the data is identical the following display will result:

```
COMP MAY/DAT:3 :4

COMP - Version 2.0.0 - file or disk compare program
Copyright 1982/88 MISOSYS, Inc., All rights reserved

MAY/DAT:3    contains    17 sectors, EOF offset =   70
MAY/DAT:4    contains    17 sectors, EOF offset =   70
```

Notice that the second filespec was indicated by a drivespec. This is the ONLY exception to a complete filespec which is allowed. Since the files proved to be identical, only the number of compared sectors followed by the end-of-file offset were displayed.

In the case of differing files the following would occur:

```
COMP F82/DAT:3 F82/DAT:4 (R=4)

COMP - Version 2.0.0 - file or disk compare program
Copyright 1982/88 MISOSYS, Inc., All rights reserved

Posn= X'0005,00 F82/DAT:3 = X'20, F82/DAT:4 - X'00
    29 bytes did not match.

Posn= X'0005,B0 F82/DAT:3 - X'54, F82/DAT:4 - X'00
    32 bytes did not match.

F82/DAT:3    contains  18 sectors, EOF offset = 100

F82/DAT:4    contains  18 sectors. EOF offset = 100
```

The display shows record number of a discrepant sector followed by the relative byte, and the contents of that byte in each filespec. The second line displays the total number of subsequent bytes which do not match. If the ALL parameter had been specified, each of the sixty-one bytes would have been displayed in the first format revealing the content of both files.

COMP – File or Disk Compare

To compare one disk to another, use drive numbers instead of filespecs. The starting cylinder and sector number may be specified either in X'00' format or as a decimal integer. The number of contiguous sectors to compare may also be specified by using the NUM= parameter.

Unlike file to file comparisons, the disk to disk compare will only display the utility name and return to LS-DOS Ready if no divergent bytes are detected. If discrepant bytes are detected the following will appear on the video:

```
Cyl X'0D, Sec X'00, Byte X'00,
Drive 2 = X'6D, Drive 3 = x'31

3078 bytes did not match.
```

If the ALL parameter had been specified then each different byte would display in the first line format. To send the output to the printer as well as the video, specify the PRINT parameter.

FASTBACK and FASTREAD

FASTBACK is a utility program that will copy a file larger than can be held on a single floppy disk to multiple diskettes. FASTREAD will restore a set of floppy diskettes back to a file on a hard drive. These programs are written entirely in machine language, and provide a considerable time saving when compared to the standard Tandy file backup programs written in BASIC. Both of these programs may be started up and run automatically with a JCL file to allow automated backup procedures to be used.

Installation

As with any computer software product, it is recommended that you make a backup copy before starting to use the package. Do so, and place the original master in a safe place. To install the programs on the hard disk, do a backup from the floppy onto the hard drive (normally, onto the partition used as drive 0).

FASTBACK - Hard disk to floppy

Prior to using FASTBACK, you need to determine how many floppy disks will be needed to hold the file you will be copying. Single sided 40 track disks will have a capacity of 171K; double sided disks will have 342K. Format all of the disks needed, and then FORMAT 2 EXTRA. Label all disks except the extra disks with the name of the file, and number them consecutively from 1 to the end. Put labels on the extra disks but do NOT number them at this time. These extra disks can be used in the event that one of the numbered disks develops a flaw during use. This will be explained later.

Any disks that have locked out tracks CANNOT be used by FASTBACK. If tracks do get locked out during the formatting procedure, set the disk aside (or throw it away) and format another in its place. Remember - you are making a copy of the file in case something happens to the original on the hard disk. Why take chances with marginal floppy media?

To start the file save procedure, place the first floppy disk in the drive to be used and type in the command FASTBACK at the DOS Ready prompt. Three prompts must now be answered to start the file save. The first prompt asks whether you want to verify all writes to the floppy disk:

```
        VERIFY all writes (Y/N) ?
```

Normally, this prompt will always be answered Y (for Yes). In the event that time is extremely critical (approaching electrical storm, office closing in 20 minutes, etc.), you can answer this N (for No), and the file copy time will be cut approximately 50%. This is NOT recommended, however, and should only be done in emergency situations and with known reliable floppy drives.

The second prompt will be for the name of the file on the hard drive you wish to save to floppy:

```
        Source file ?
```

Type in the name of the file, including the extension (if any) and the drive number. If the file is not found, the message "File not in directory" will appear and the program will exit back to the DOS Ready prompt. Once the source file is found and opened, the next prompt will be for the floppy drive number to save it to:

```
        Destination drive ?
```

Answer this with the drive number holding the floppy disk. The file save will now begin, and two lines of information will appear on the screen:

```
        Using disk # n
        Writing cylinder nn
```

The disk number shown should correspond to the number of the disk in the floppy drive. If it doesn't, check to see that all the disks in the floppy set you are using are numbered consecutively and are in order. At any time, pressing the BREAK key will abort the file save and return to the DOS Ready prompt. Once the disk is full, you will be prompted:

```
        Insert disk N, Press ENTER
```

The "N" will be the number of the next disk in the set. Place it in the floppy drive and press the ENTER key to resume copying. This procedure will continue until the file is entirely saved to floppy. When finished, the screen will clear, the message "Completed successfully" will be displayed, and the DOS Ready prompt will appear.

FASTBACK error messages

There are certain error messages that can appear if FASTBACK detects something wrong. Three of them can appear even before the program actually starts writing to floppy:

```
        Not enough free memory for FASTBACK

        Destination disk MUST be a floppy drive

        Source & Destination disks can't be the same
```

FASTBACK requires approximately 8K of free memory after it is loaded. Since the program itself is less than 3K long, this message should NEVER appear. If it does, it indicates a severe hardware error or a bad configuration at the DOS level. Stop immediately and check things out!

The other two errors usually come from mis-typing the drive number on the source filename or for the destination drive. Check your answers and restart the program with the proper values.

One error message deals with flawed tracks on a disk:

```
        Disk has locked out tracks. Press ENTER
```

If you see this message, the floppy disk you just inserted had tracks locked out during formatting. As mentioned earlier, this is a no-no. Press ENTER, and the original prompt for the floppy disk will re-appear. Pull out the offending disk and put in a new one. If the disk in question is one of your numbered set, take one of the extra disks, number it to replace the flawed one, and insert it instead.

This brings us to the two last error conditions - an error when writing to the floppy, and an error when reading from the hard disk:

```
        Floppy disk error

        Put in replacement disk, press <ENTER>

        DO NOT press Break unless you want to quit!!
```

This message will appear anytime there is an error writing or verifying the floppy disk. Here is where the extra disks come into use. Pull out the bad floppy, take an extra disk and number it to match the bad disk, place it back in the drive and press ENTER. If you want to abort the copy instead (or if you didn't make those extra disks), press BREAK to quit and return to DOS Ready.

The hard disk read error will show as:

```
Source disk read error, <BREAK> to abort,
<ENTER> to ignore
```

If this message appears, there is a bad sector on the hard disk. The choice is yours - continue and get as much of the good information as possible, or stop the file save. Press the indicated key once you decided what to do.

Automated use from JCL

To run FASTBACK from a JCL file, you need a minimum of four lines:

```
FASTBACK
Y
FILENAME
DRIVE#
```

Line I must always be the command FASTBACK to execute the program. Line 2 is usually "Y" (Yes to the verify question). Line 3 is the name and drive of the source file. Line four is a single digit for the floppy drive #. Of course, comment lines giving information can be included as shown in the next example. Also, since all keyboard responses inside FASTBACK are done in a "non-JCL" mode, two or more of these sequences can be put together in the event that there is more than one file to save. The following example is one method of starting up FASTBACK:

```
. SEMI-WEEKLY backup of mailing list
. Get "A" set of disks
. Place disk 1 in drive 5 (top floppy)
. Close door, and press ENTER when ready...
//pause
FASTBACK
Y
MAIL/KEY:1
5
```

FASTREAD - Restore floppy to hard disk

FASTREAD will read a set of floppy disks created with FASTBACK and restore the original file back to the hard drive. The hard drive partition does not have to be the same one the file came from originally.

To start the program, type the command FASTREAD at the DOS Ready prompt. Several prompts will now appear. The first is for verification of the writes to the hard drive:

```
        VERIFY all writes (Y/N) ?
```

This prompt should always be answered by pressing Y (for Yes). The time saved for not verifying will be approximately 5 seconds when restoring from a 40 track, single sided floppy. In extreme conditions, there may be a reason to not verify the writes, but it is not recommended. The next prompt will be for the drive holding the floppy disk:

```
        Source drive ?
```

Be sure the first floppy disk is in the drive and the door is closed, and respond by pressing the appropriate number. At this point, the filename of the original file will be read from the floppy and displayed on the screen.

The next prompt will be for the hard disk to receive the file:

```
        Destination drive ?
```

Answer with the hard disk drive number where the file should go. The restore will now start and the screen will show two messages:

```
        Using disk # n
        Reading cylinder nn
```

The disk number currently being read will be shown, and should match the label on the disk. The cylinder number will increase from 1 until the floppy disk has been entirely read. The next disk will be called for with the prompt:

```
        Insert disk N, Press ENTER
```

The "N" represents the number of the next floppy disk in the set. Insert it and press ENTER. Pressing BREAK at this point or during the reading will abort the restore and return to the DOS Ready prompt. However, be aware that the file on the hard drive will be changed once FASTREAD has started.

The disks will be prompted for until the last disk is read. At that point, the screen will clear, the message "Completed successfully" will be displayed, and the DOS Ready prompt will appear.

FASTREAD error messages

Disk read/write errors, trying to restore disks out of order, or using non-FASTBACK floppy disks will all show different error messages on the screen. FASTBACK writes certain information to an otherwise unused portion of the floppy disk for FASTREAD to check during the restore. If this information is missing, or if disks from different FASTBACK sessions are mixed, an error message will also appear. Three FASTREAD generated errors can appear at the start of the restore:

```
Not enough memory for FASTREAD

Not disk number 1. Insert correct disk, press <ENTER>

Not a FASTBACK disk - can't read
```

FASTREAD requires approximately 8K of free memory after it is loaded. Since the program itself is less than 3K long, this message should NEVER appear. If it does, it indicates a severe hardware error or a bad configuration at the DOS level. Stop immediately and check things out! The next error message indicates that the floppy disk in the set is NOT the first disk in the set. Find the correct disk, place it in the drive, and press ENTER to continue. Pressing BREAK will abort the restore at this point. The last message indicates that the floppy disk was not created by FASTBACK. The usual cause of this is typing in the wrong source drive number. Check your responses and re-start the procedure.

Two error messages can appear if errors on the source floppies are detected:

```
Error reading header. <BREAK> to abort,
<ENTER> to retry

Source disk read error. <BREAK> to abort,
<ENTER> to ignore
```

The first message will appear if there is a disk read error when a floppy is initially being read. You should retry several times by pressing ENTER, because the header information MUST be read successfully before the data on the disk can be restored to the hard drive. If the retries are not successful, pressing BREAK will exit back to DOS. The second error indicates a read error in the data saved on the floppy. This is usually caused by something that happened to the floppy after it was created with FASTBACK - exposure to a magnetic field, touching the floppy media with fingers, etc. Pressing enter will ignore the error and attempt to read the next sector on the disk. Pressing BREAK will abort the restore and return to DOS. This message may appear several times in a row if more than one sector of the disk is bad.

Two error messages can appear when accessing the hard drive:

```
Disk write error. <BREAK> to abort, <ENTER> to ignore

Verify error. <BREAK> to abort, <ENTER> to continue
```

Since the hard drive media cannot be physically changed, errors when writing or verifying can be ignored so the entire file is copied. However, it might be a good idea to re-format the hard drive so the bad areas are locked out, and then restore the data to it.

Automated use from JCL

Like FASTBACK, the FASTREAD program can be started with a JCL file. Again, four lines are necessary - the FASTREAD command, the Verify answer, and the source and destination drive numbers.

For example:

```
FASTREAD
Y
5
1
```

This example JCL file would initiate FASTREAD, use verify, read from floppies in drive 5, and write to the hard drive in logical drive position 1.

FED2

Utility Overview

FED2 allows the user to access a disk file to display, disassemble, and edit that file. It is a screen-oriented File EDitor to be used with a LS-DOS compatible operating system. Its wide range of capabilities make it an excellent tool for the advanced user. Its simplicity makes it easy to use for the novice.

FED2's main features are as follows:

1) Substitution editing capabilities are supported. The user can easily position to any byte in any given record. Hexadecimal and ASCII modification are available. Direct disk patching becomes a simple matter with FED2. Small Changes in files can be made quickly. With FED2, there is no need to reassemble large source files merely to change one byte.

2) FED2 allows record advance, backspace, and absolute positioning. Paging back and forth through the file is accomplished at a keystroke. The user does not need to know any diskette information (such as density, number of sides, number of sectors per gran, etc.). The required information necessary to start FED2 is the file name.

3) ASCII, literal text or hex string searches are easily performed. A repeat command exists to position to subsequent occurrences of the same string. FED2 searches the entire file, not just the current record. It searches for text or ASCII strings up to 16 characters in length. Searching for Hex strings of up to 6 bytes in length is supported.

4) Mapping of machine language (/CMD) files with loader code blocking and Z-80 disassembly is available for LS-DOS load

module format files. The user can page through each load block (forward or backward), or position to the byte in the file which loads at a specific address. It is also possible to position to the next Z-80 instruction or position to the address referenced by the current instruction. These features allow the user to step through and examine machine language routines within a file. Direct patches are made quickly and easily.

5) Complete listing of a file, individual records, and disassembly output of load module files to a printer are supported.

6) FED2 includes a standard 256 byte display mode, and FED2 6.x includes a display for files with record lengths other than 256. Under 5.x, a screen mode is available to display additional information not available in the 256 byte display mode due to lack of video space.

7) A Disk Mode is also available to work with an entire disk (at the cylinder/sector level).

Information on disk organization, file structure, and load module format files can be found in the Appendix of this manual section.

Throughout this manual, a character or word between angle brackets is used to represent a keyboard key. Thus the symbol, <ENTER>, refers to the keyboard key marked ENTER and not a five letter word. <P> means the "P" key etc.

Entering FED2

The full syntax for invoking FED2 is:

FED2 – Disk and File Editor

```
FED2 [*][!][filespec[,lrl]][drivespec]
```

`*` an optional prefix used to force FED2 not to map a command file.

`!` an optional prefix used to suppress FED2's automatic match of the filespec to DOS library members.

`filespec` an optional file specification designating the file you wish to edit.

`lrl` an optional parameter to specify a logical record length other than 256 for the file.

`drivespec` an optional drive specification of the drive you wish to edit.

If a load error occurs, refer to the FED2 Load Errors section. Notice that entering a file or drive specification on the command line is optional. Once FED2 has loaded, a prompt will appear for the filespec if none was entered on the command line. Answer this prompt by giving the file specification or drive specification you wish to examine or modify. The filespec must be entered using the following syntax:

```
        *!Filename/ext.password:drivespec,lrl
```

The leading asterisk is an optional parameter (normally not used) which is used to inhibit FED2's mapping of executable command files. This is necessary if you want to search a command file while regarding load module record type data as significant. There will be more information on this later.

FED2 will examine the entered filespec to see if it matches one of the DOS library commands. If so, FED2 will access the appropriate DOS system file which contains that command. If the file you wish to edit has the same name as a DOS library command, FED2 will assume you wish to edit the library file. You can keep FED2 from matching your filespec with library names by prefixing the filespec with an exclamation point.

The filename may consist of up to 8 alphanumeric characters, the first of which must be alphabetic. All filespecs must contain at least the filename. The extension may consist of up to 3 alphanumeric characters. Like filenames, the first character must be alphabetic. The extension is optional. The default extension for all filespecs is "/CMD". To enter a file which has no extension, follow the filename with a slash "/". The optional password has the same form as a filename. The password is necessary only if the protection level is EXEC or higher. If the file has *READ* access, then FED2 will not allow writing to the file unless the owner password was given. The drivespec is a colon followed by a number between 0 and 7 which is a working drive number. This is an optional field. If no drivespec is entered, all active drives in the system will be searched for the filespec. The first match found will be used. If the drivespec is used by itself, the entire disk will be treated as a file (see Disk Mode).

LRL is a number ranging from 1 through 256. Like the drivespec, the LRL is optional. If no LRL is specified, the default value will be 256. You would normally use the LRL parameter only if you wish to examine a file using its logical record length and it is other than the typical 256.

To exit FED2 at this point rather than entering a filespec, press the <BREAK> key, and control will return to the DOS level. If an illegal or improper filespec is given, the appropriate error message will appear, and the filespec prompt will re-display.

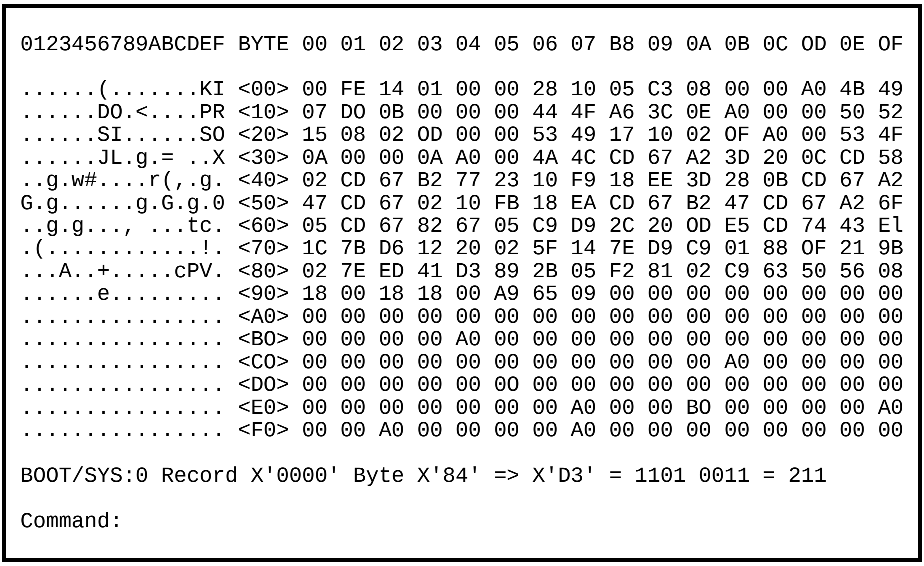
It is advised that when using FED2, the <BREAK> key should always remain enabled. The <BREAK> key is necessary to abort any operation and to terminate others. Because of this, FED2 will always enable the <BREAK> key when it is invoked.

Hexadecimal notation (X'nn') will be used to represent the current record number and relative byte number. After a valid filespec has been given, record X'0000' will appear on the screen, and be resident in the "edit buffer". The term "edit buffer" will refer to the record of the file currently in the computer's memory which is simultaneously being displayed. The edit buffer (also referred to as current record) will contain one record (1-256 bytes) at any given time. There will be two cursors flashing within the record; one cursor will be in the "ASCII" portion of the screen, the other cursor will be in the "Hex" display portion. Upon initially accessing a file, these cursors will be positioned over relative byte X'00' of record X'0000'. Throughout this documentation, the term "relative byte" will be used, and will indicate the byte number (X'00'-X'FF') relative to the beginning of the current record.

There will be an input cursor located on the lower portion of the screen following the message "Command". This will be referred to as the "command buffer", and will be used to pass commands to FED2.

Additional information shown on the screen will be the current record number, filespec, relative byte within the sector, etc. The following sample displays show where this information will be presented.

Display mode under FED2



```
0123456789ABCDEF BYTE 00 01 02 03 04 05 06 07 B8 09 0A 0B 0C OD 0E OF

......(.......KI <00> 00 FE 14 01 00 00 28 10 05 C3 08 00 00 A0 4B 49
......DO.<....PR <10> 07 DO 0B 00 00 00 44 4F A6 3C 0E A0 00 00 50 52
......SI......SO <20> 15 08 02 OD 00 00 53 49 17 10 02 OF A0 00 53 4F
......JL.g.= ..X <30> 0A 00 00 0A A0 00 4A 4C CD 67 A2 3D 20 0C CD 58
..g.w#....r(,.g. <40> 02 CD 67 B2 77 23 10 F9 18 EE 3D 28 0B CD 67 A2
G.g......g.G.g.0 <50> 47 CD 67 02 10 FB 18 EA CD 67 B2 47 CD 67 A2 6F
..g.g..., ...tc. <60> 05 CD 67 82 67 05 C9 D9 2C 20 OD E5 CD 74 43 El
.(............!. <70> 1C 7B D6 12 20 02 5F 14 7E D9 C9 01 88 OF 21 9B
...A..+.....cPV. <80> 02 7E ED 41 D3 89 2B 05 F2 81 02 C9 63 50 56 08
......e......... <90> 18 00 18 18 00 A9 65 09 00 00 00 00 00 00 00 00
................ <A0> 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................ <BO> 00 00 00 00 A0 00 00 00 00 00 00 00 00 00 00 00
................ <CO> 00 00 00 00 00 00 00 00 00 00 00 A0 00 00 00 00
................ <DO> 00 00 00 00 00 0O 00 00 00 00 00 00 00 00 00 00
................ <E0> 00 00 00 00 00 00 00 A0 00 00 BO 00 00 00 00 A0
................ <F0> 00 00 A0 00 00 00 00 A0 00 00 00 00 00 00 00 00

BOOT/SYS:0 Record X'0000' Byte X'84' => X'D3' = 1101 0011 = 211

Command:
```

FED2 LIBRARY

<A>	Enter ASCII modification mode
<B>	Position to the Beginning of the file
<C><ENTER>	Clear record with zeroes
<D><ENTER>	Disassemble file to printer
<E>	Position to the End of the file

<F>	Enter Find mode and:
<A>	find ASCII string
<H>	find Hexadecimal string
<T>	find Text string
<L>	find load address
<G>	Go to the next occurrence of last search
<H>	Enter Hex modify mode
<I>	Position to next Z-80 Instruction
<J>	Jump to current instruction reference
<L><ENTER>	List disk rite to printer
<N><ENTER>	Enter a New file
<P><ENTER>	Print current record in edit buffer
<R>	Position to Record
<S><ENTER>	Save current record in edit buffer
<U><ENTER>	Update file's directory entry
<X><ENTER>	Exit FED2 and return to DOS Ready
<BREAK>	Cancel current FED2 command
<ENTER>	Display FED2 instruction set (Menu)
<;> <+>	Advance one record in the file
<-> <=>	Backup one record in the file
<<>	Position to previous load block
<>>	Position to next load block
</>	Toggle between Directory entry and HIT position

Note: FED2 is an advanced utility, giving the user an opportunity to accomplish tasks not easily performed by other means. FED2 can also be a hazard to the inexperienced or uninformed user. It is strongly recommended that all editing be done on a BACKUP copy of the original file or disk whenever possible.

Cursor Movement Commands

<@> nn	Position cursor to relative byte X'nn'
<Lt arrow>	Move cursor left one byte
<Rt arrow>	Move cursor right one byte
<Up arrow>	Move cursor up one line
<Dn arrow>	Move cursor down one line

<SHIFT><Up arrow> Position cursor to relative byte X'00' of the current record

<SHIFT><Rt arrow> Position cursor to end of the line
<SHIFT><Lt arrow> Position cursor to start of the line

The @ positioning command requires a hex byte consisting of two hex digits. This may be any number X'0' through X'FF'; however, if a single digit is used (no leading zero), the <ENTER> key must be pressed in order to execute the command. This is because the <@> command expects two digits. If two digits are used, the command will execute immediately after the second one is typed.

This arrangement is used throughout FED2 whenever information must be supplied in addition to the command key. Merely remember to press <ENTER> if nothing occurs after typing in a command. This is especially true for the strings used in all of the FIND subcommands.

The arrows may be used from within either the ASCII or Hex modification modes to position within the buffer. The "@" sign will not be accepted during either modify mode.

The cursor movement commands will not wrap into either a prior or a subsequent record. To switch records, use the record manipulation commands. No cursor movement will occur if an attempt to violate buffer boundaries results.

Record Manipulation Commands - File mode

<+>

Advance one record sequentially in the file. For example, if the current record is X'000C', after pressing <+>, record X'000D' would be displayed (provided that it exists). An "*" will be displayed adjacent to the record number when positioned to the last record in a file. Issuing the <+> command will not change the position of the relative byte cursors. A "+" will be shown in the command buffer to indicate forward motion in the file.

<->

Back up one record sequentially in the file. If the current record is X'0087', after pressing <->, record X'0086' would be displayed. Issuing the <-> command does not change the position of the relative byte cursors. The <-> command will be ignored if it is issued when record X'0000' is being displayed. A "-" will be shown in the command buffer to indicate retrograde motion in the file.

<B>

Position to the beginning of the file (record X'0000') and position cursors to relative byte X'00'.

<E>

Position to the end of the file. An "*" will appear adjacent to the record number, indicating that the record being displayed is the last record in the file. If the file has an LRL of 256, then the relative byte cursors will be positioned on the last byte in the file which is often referred to as the "End-Of-File offset byte". (Note that this is not necessarily relative byte X'FF'.) For LRLs other than 256, the cursors will be positioned at the last byte in the record. Any modifications made to bytes beyond the EOF offset byte are usually superfluous.

<R>nnnn

Position to Record X'nnnn', provided record X'nnnn' exists in the file. If the record does not exist the request will be ignored. After entering <R>, the prompt Record X' ' will appear in the command buffer. The input for the record number will be taken within the single quotes. Hex digits (0-F) must be entered. Any other characters will be ignored. <BREAK> will cancel this command. The user may enter the record number without using the standard four digit (X'nnnn') format. Simply type in the record number and press <ENTER>. For example, if the desired record number is X'0021', type <H><2><1><ENTER>. To position to record X'0007', type <R><7><ENTER>. The position of the relative byte cursors will remain unchanged after positioning to the new record.

</>

This command either positions from a byte in the HIT (Hash Index Table) to a directory entry, or from a directory entry back to the HIT. If the cursor is positioned somewhere in the HIT, </> would reposition to the directory entry corresponding to that HIT position. If positioned in a directory entry, </> would reposition to the byte in the HIT corresponding to that entry. This command is only applicable while in DIR/SYS (file mode) or in the directory cylinder (disk mode). It serves no purpose elsewhere and does not function in any other file or any other cylinder. For more information, consult your DOS manual on DIRECTORY structure.

FED2 Modification Commands

<A>

Enters the ASCII Modification Mode. In this mode, modifications can be made directly in ASCII. Any character that can be generated from the keyboard (with the exceptions of the <BREAK> key and the arrow keys) can be directly entered into the edit buffer. Modifications can be made by positioning the cursor over the bytes to be changed. After the <A> command is issued, the cursors will become larger and the command buffer will display "ASCII Modify". From this point on, any characters entered will be taken as modifications to the bytes in the edit buffer. After a character is entered, the cursors will position to the next character in the edit buffer. If the cursors are positioned at the last character in the edit buffer, then no advance will occur. The arrow keys may be used to position the cursor without altering the buffer contents. Any changes made to the buffer WILL NOT automatically write to the file. In order to save changes to the file, see the <S>ave command. To exit the ASCII modify mode, press the <BREAK> key.

<H>

Enters the Hexadecimal Modification Mode. In this mode, modifications to bytes in the edit buffer are accomplished by typing hexadecimal digits. After the <H> command is issued, the cursors will become larger and the command buffer will display "Hex Modify". From this point on, hexadecimal digits (0-F) must be entered to modify bytes in the buffer. Note that since a single byte is represented by two hex digits, hex modify edits one nibble (half a byte) at a time. The arrow keys may also be used to position the cursors for additional editing. To exit the Hex modify mode, press the <BREAK> key. Like the ASCII Modification Mode, no changes are automatically made to the file. To make the modifications to the file, see the <s>ave command.

<C><ENTER>

Clears the buffer contents from the cursor position to the end of the buffer by filling it with X'00'. Some files have erroneous or random information past the end-of-file offset byte in the last record. The clear command can be used to overwrite the remainder of the buffer with zeros to facilitate viewing. Again, since none of the edits perform an automatic write to disk, <S> is necessary to save the buffer contents.

<S><ENTER>

Save the contents of the current edit buffer to disk. The current record will overwrite the contents of the disk record. Although the changes are made, neither the date nor the mod flag of the file in the directory record will be changed.

<U><ENTER>

This command is used to update the modification date in the directory entry of the current file being edited. The modification flag will also be set. Update only works in the file mode. The current date will be displayed in the prompt:

```
      Update directory (mm/dd/yy) ?
```

If you wish to set the directory date for the file to this date, just depress the <ENTER> key. If you wish to choose another date, enter it in the form "mm/dd/yy". Your date will be checked for validity prior to updating the directory. If the date is not acceptable, the UPDATE command will abort. If the disk containing the file is using the extended dating convention, then the extended date field will be set and the time field will be set to "00:00".

Note that updating the directory date is normally not done by FED2 even if the file has been altered via the <S>ave command.

FED2 Control Commands

<N><ENTER>

Causes a prompt for a new filespec. FED2 will clear the screen, print its sign-on message, and prompt the user for a new filespec to be edited. Note that strings are saved so that <G> will work on the new file without reentering a search criterion.

<X><ENTER>

Exit to operating system. Any changes made to the current record buffer, and not written to disk with the <S> command will be lost.

<ENTER>

The <ENTER> key is used as a confirmation to complete most commands that result in significant alteration of the current record. <ENTER> alone will display a menu containing most of the FED2 commands, and a brief description of their use. Pressing <ENTER> at the menu page will return to the display mode.

<BREAK>

The <BREAK> key is used to abort a FED2 command in progress.

FED2 Output Commands

<P><ENTER>

Print the current buffer contents, to a printer. If the printer is not available, the error message *"Printer Not Ready"* will display. Pressing <ENTER> at the error will attempt to print again. Pressing <BREAK> will abort the operation. Under DOS 6.x, other error messages are possible if the printer is routed or linked to a disk file or a device.

<L><ENTER>

List the file to a line printer starting with the current record. The record length must be 256. Since FED2 does its own pagination, it is suggested that no printer filters involved with line counting be used in conjunction with this command. The listing will terminate when the end of the file is encountered, or when <BREAK> is pressed. Error handling works exactly as with the <P> command.

<D>

Output disassembly of a Load Module File starting with the current instruction to a line printer. If the cursor is positioned in loader code, the first instruction following will be used. The listing will terminate when the end of the load module is encountered, or when <BREAK> is pressed. Unlike the <L> command, there is no pagination on the disassembly output. Error handling is identical to the <P> command.

FED2 Search Commands

<F>

Enters the FIND mode. A "FIND" prompt will appear in the command line. A subcommand declaring the type of search must be entered. After entering the subcommand, an appropriate prompt will display followed by a blank space in quote marks. The following are the four FIND sub-commands:

<A>

Find ASCII "string". This is a literal search for the exact ASCII characters entered. "string" is a group of from one to sixteen ASCII characters. The only ASCII characters that can't be generated are the <ENTER>, <BREAK> and <BACKSPACE> keys. If less than 16 characters are typed, the <ENTER> key must be pressed to initiate the search.

<H>

Find Hexadecimal "string". This is a literal search for the exact hexadecimal bytes entered. "string" is a group of up to 12 hex digits (6 bytes). Only the valid hex characters (0-F) will be accepted.

<T>

Find Text "string". This is a search for ASCII characters that ignores differences in upper or lower case. The same restrictions which apply to <A> apply to <T>.

<L>

Find Load Address X'nnnn'. This is a search for the byte in the file which loads at memory location X'nnnn'.

<G>

Go to the next occurrence of the last searched string or load address. In order to do the same search again, it is necessary to use this command. It simply looks for the last item specified again. It also "memorizes" the last search criterion as long as FED2 is active. This means that searches through different files for the same string are possible without re-entry of the string. The only exception to this is an attempted <F><L> when switching to a non-load module format file. Obviously the string is useless in that case. If it is not obvious then read the appendix. If it is still not obvious just take our word that it makes no sense.

Pressing <BREAK> at any time during the input sequence or actual search, will cause FED2 to display the record which was current before the search started. Any changes made to the current edit buffer and not saved will be lost during a search. The <BACKSPACE> key may be used to correct any mistakes made during input.

During a search, the record number being searched will be displayed. If the string or load address is not found, the appropriate message will be displayed. Control will return back to the position in the file before the search. Note : A search starts at the byte following the current position. If the cursor were positioned to relative byte X'FF' of record X'0012', the search would start at relative byte X'00' of record X'0013'. For load module format files, only data found in object code blocks will be used in the search. Characters in header, comment, filename or any other blocks will be ignored. It is also assumed that object code blocks load contiguously in memory.

In order to search a Load Module File and include load blocks, precede the filespec prompt (the very first thing FED2 asks for) with an asterisk

FED2 Load Module Commands

<I>

Position to the next Z-80 instruction. For example, if the cursor is positioned at a CALL instruction, and the <I> command was used, the cursors would be positioned 3 bytes after the X'CD' opcode. If the instruction spans a load block, an additional 4 bytes will be skipped. Note: Using the <I> command may position to the next record in the file. If this should happen, any changes made to the edit buffer which were not saved will be lost. If the cursor is positioned in load code, <I> will position to the first valid instruction found. Note that opcodes out of sequence will be just as readily disassembled. Make sure the "logic thread" being followed is the correct one.

<J>

Position to Z-80 instruction reference. This command positions to the address operand of the current instruction. For example, if the current instruction was a JP 67A9, issuing a <J> would attempt to position to the byte which would load at address X'67A9'. If the address is not located in the file, the error message "Load Address not Found" will be displayed, and the original cursor position will be restored. The <J> command may be used for any instruction referring to an absolute address or relative branch (CALL, JP, JR, LD, DJNZ), whether conditional or unconditional. If the cursor is either at the end of the module or with the transfer address block, a <J> will locate the transfer address. Any address used by <J> must be a 16 bit address or a JR offset. Branches such as eight bit loads, JP (HL) or RST will not work. For example, the instruction LD A, 5 will not attempt to locate X'0005', LD HL, 6060 will, however, attempt to locate X'6060'. Note: any changes made to the edit buffer prior to using the <J> command will be lost if the current record is left, unless saved with the <S> command.

">" "<": Right and Left angle brackets

Positions the cursors to the next/previous loader block (X'01', X'02', X'03', X'05', X'07', X'10', X'1F') of a Load Module File. This feature was designed to allow the user to trace through machine language files quickly. Encountering a X'02' will terminate a trace. For more information on "Type" bytes, refer to the appendix on *LOAD MODULE FORMAT FILES.*

Disk Mode

Occasionally it is desirable to work with an entire disk as opposed to a single file. For this purpose, the FED2 disk mode makes it possible to treat the entire disk as a file. Most of the commands available in the file mode are also available in the disk mode. To enter the disk mode, simply give the drivespec (colon followed by a number between 0 and 7) at the "Filespec:" prompt or on the command line.

Since the entire disk is treated as a file, positioning to specific cylinders and sectors is accomplished by specifying the record number.

The following commands illustrate the difference. More information may be obtained from the appendix on disk organization.

<R> ccss

Position to Record X'ccss' on disk. The Record number consists of the cylinder number (cc) and sector number (ss). For example, positioning to cylinder X'34', sector X'05' (record X'3405'), would be accomplished by typing <R><3><4><0><5>. Pressing <ENTER> following the record number is required only for requests consisting of less than four digits. To position to Record X'030A', type <R><3><0><A><ENTER>. To position to Record X'0004' type <R> <4><ENTER>.

<B>

Position to Beginning of Disk (Cylinder 0, Sector 0). After issuing a <B>, the current Record number would be X'0000'.

<E>

Position to End of disk. The actual record number would depend on the type of disk. A single-sided double density 40-track diskette would have an ending record of X'271 1' (Cylinder X'27', Sector X'11').

<+>

Position to next record. If positioned at record X '1405', (Cylinder X'14', Sector X'05'), after pressing <+>, record X'1406' (Cylinder X'14', Sector X'06') would be displayed. If the <+> is used when positioned at the ending sector number of a cylinder, sector 0 of the next cylinder would be displayed. For example, a double-density, double-sided, five-inch diskette has 36 sectors per cylinder (numbered from X'00' - X'23'). If the current record is X'0223', after issuing <+>, record X'0300' would be displayed.

<->

Position to previous record. If positioned at record X'1405', (Cylinder X'14', Sector X'05'), after pressing <->, record X'1404' (Cylinder X'14', Sector X'04') would be displayed. If the <-> is used when positioned at Sector B of a cylinder, the ending sector of the previous cylinder would be displayed. For example, one configuration for a 5" hard disk might use one platter for a logical drive. Each cylinder might contain 64 sectors (numbered x'00 - X'3F'). Issuing a <-> when positioned at Record X'5000' (Cylinder X'50', Sector X'00') would cause record X'4F3F' (Cylinder X'4F', Sector X'3F) to be displayed.

Another feature of the DISK MODE is current file indication. Under 5.x, this is shown in the 128 byte display mode. When positioned to a sector on a disk which is allocated to a file, the filespec along with the relative record number is displayed. With this feature, reconstruction of a damaged directory is possible.

If FED2 attempts to access a disk which it cannot distinguish, the error message "Can't Log in Disk" will be displayed. Like other errors, pressing <BREAK> will cancel the current command, causing the filespec to reprompt. Pressing <ENTER> will indicate to FED2 that it should use the information in the DCT (Drive Code Table) to access that disk. If FED2 cannot interface properly, it may be necessary to use DEBUG or another utility to fix the disk.

Practical examples of FED2's use

1) Change the byte which loads at address X'57CE' to an X'C9', in a file named TEST/CMD on drive 2:

A) To edit the file, at the Filespec prompt type: **`TEST:2<ENTER>`**

B) To position to address X'57CE', type: **`<F><L>57CE`**

C) To enter hex modify mode, press: **`<H>`**

D) Then type in the change: **C9**

E) To exit the hex modify mode, type: **`<BREAK>`**

F) To save the change to disk, type: **`<S><ENTER>`**

G) To exit FED2, type: **`<X><ENTER>`**

Note that in this case, no extension was required to access the file, because the default extension of /CMD was correct.

2) To null out record X'7A' of a data file named ACCOUNTS/DAT which has an update password of BOSS:

A) To edit the file, at the filespec prompt type:
`ACCOUNTS/DAT.BOSS<ENTER>`

B) To position to record X'7A', type: **`<R>7A<ENTER>`**

C) To fill the buffer with zeroes, type: **`<C><ENTER>`**

D) To save the changes to disk, type: **`<S><ENTER>`**

E) To exit FED2, type: **`<X><ENTER>`**

3) To change the string "Burgers" to "Hot Dog" in a file named THEMENU/SCR on drive 6, after editing a different file:

A) To enter a new file, type: **`<N> <ENTER>`**

B) To edit the file, type: **`THEMENU/SCR:6<ENTER>`**

C) To find the ASCII string "Burgers". type:
`<F><A>Burgers<ENTER>`

D) To enter the ASCII modify mode, type: **`<A><ENTER>`**

E) To change the ASCII string to "Hot Dog", type: **`Hot Dog`**

F) To exit the ASCII modify mode, type: **`<BREAK>`**

G) To save the changes to disk, type: **`<S><ENTER>`**

H) To exit FED2, type: **`<X><ENTER>`**

4) To find out the name of a file on drive B containing the string "joe dude" (in either upper or lower case):

A) To access the disk as a file, enter the drivespec: **`:0`**

B) To find the text string, type: **`<F><T>joe dude<ENTER>`**

C) Under 5. 1. enter the 128 byte display mode by typing: **`<M>`**

D) The filename/ext and relative record containing the string will be displayed at the lower portion of the screen.

5) Find the load address which contains the sequence of bytes X'45', X'22', & X'77' in a file named HELPME/DCT:

A) To access the file, type: **`HELPME/DCT<ENTER>`**

B) To find the hexadecimal string X'452277', type: **`<F><H>452277<ENTER>`**

C) The load address displayed refers to the first byte in the string.

Disk I/O Errors

As with any hardware, there is always that chance of something going wrong. This could happen when reading from or writing to a disk. For some reason, some component failed to do its job. The problem could be in the disk media, the disk drive, the disk controller, or the computer. Whenever an I/O error occurs under LDOS, an error number is returned to the program that requested the disk I/O function. FED2 reports the error and allows the user to decide what to do about it. There are two options available in FED2:

1) Abort the process, and resume whatever was done prior to the I/O error

2) Ignore the error, and continue the process. Pressing <BREAK>, indicates an abort operation. Pressing <ENTER> will ignore the error, and continue the process.

Error Examples

Example #1

Assume that a record from a file on a write protected disk was read in. After examining the record, some changes were made to the edit buffer. Once the operator was content with the changes, the <S>ave command was issued. Almost immediately, the error message "Write Protected Disk" is displayed. To make the changes, simply take off the write protect tab and re-issue the <S>ave command.

Example #2

While paging through a file, the error message "Parity Error During Read" is displayed, The edit buffer contains the record which had the error, however the integrity of the data is doubtful. At this point, the edit buffer could be modified and followed by a <S>ave attempt.

For a detailed description of I/O errors, refer to the Operating System manual.

FED2 Load Errors

Two rare errors need to be examined. If while attempting to execute FED2, the error message "Insufficient Memory to load FED" appears, one of two undesirable events has occurred.

1) The amount of memory left in the system is not enough to execute the program, or

2) Loading FED has overwritten reserved memory (HIGH$). Because of the danger of #2, the solution is to re-boot the system. Re-enter FED2 only if more free memory is available.

The second rare error is that a file is so large and the amount of free memory so sparse that FED2 runs out of memory to map a file into load blocks. No error message will be issued but the current file will NOT be mapped. This is exactly like typing "*" for the first character of the filespec. This error is not fatal but inconvenient.

Both errors occur rarely because the amount of memory which must be reserved to cause either problem is unrealistic in normal use. However, in the interest of complete error trapping, these accommodations have been made.

Load Module Format Files

One of the most frequent tasks requested of the operating system is the system's own internal loader. Its function is to load machine language programs from load module files into memory. Everything from application programs such as FED2, to utilities like FORMAT and BACKUP, and even system files all are loaded via the system loader. All programs do not load at the same address, they seldom have the same length, and they rarely have the same execution address. Therefore, a special format was established to provide this variable information to the system loader. This is now called Load Module Format. When an LDOS compatible assembler, such as EDAS, writes an object file program (/CMD) to disk, it uses this specific format. Data is written in blocks, not necessarily contiguous.

Each block consists of.

1) 1 byte indicating the Type of block

2) 1 byte indicating the Length of this block

3) Data pertinent to the particular block type

The blocks are organized sequentially, the length of the block defined indicates the position of the next block. The system loader reads until it encounters a block type indicating that it should cease loading. The following is a list of Type bytes and functions that FED2 will acknowledge.

Type	Byte Function
X'01'	Load Object block into memory
X'02'	Get Transfer Address
X'03'	Get Transfer Address (non-executable)
X'05'	Load Module Filename Header
X'07'	Patch Header Name
X'10'	Yanked X-Patch Object block
X'1F'	Comment Block

The byte following the Type byte is a length byte, which indicates the quantity of data bytes following. The length byte has a range from X'01' to X'00' (1 to 256: note that zero is high here). A length byte of X'07' indicates seven bytes in the data field. A length byte of X'00' indicates 256 bytes of data. For Type bytes X'01' and X'10', the quantity of object code data bytes following is equivalent to the length byte minus two. This is because object code blocks contain an additional two bytes immediately following the length byte indicating the load address (See Type Byte X'01'). A length byte of X'03', indicates three bytes follow: 2 bytes for the address, and 1 byte of object code data. A length byte of X'00' indicates 256 bytes follow: 2 bytes for the address, and 254 bytes of object code data. Since the address field is always present, its length is assumed by the loader. A length byte of X'01' indicates 255 bytes (X'FF') of object code data following. A length byte of X'02' indicates 256 (X'00') bytes of object code. By subtracting two from the length byte, the actual quantity of bytes loaded can be obtained.

The data following the length byte is dependent of the type of block. There are no restrictions on what the data in the block must be. Generally, block types X'05' and X'1F' contain ASCII text, but aren't required to.

Load Object Block - Type X'01'

This type indicates that the following data is to be loaded into Memory. The two bytes following the length byte are the destination (or starting load address) of the object block. The load address is stored in LSB, MSB (Least Significant Byte followed by Most Significant Byte) format. Since two bytes of the block are used for the load address, the length byte must account for this. The easiest way to understand this is by looking at a typical block:

```
        01 08 00 70 CD 47 95 C8 B7 DD
```

The length byte indicates that this block is X'08' bytes long. The address to load the data block at is X'7000' (lsb,msb format). However, only X'06' bytes are loaded into memory. This is because two bytes of this block were used to designate the load address. Therefore the actual area of RAM that will be loaded is X'7000' - X'7005'. After this block is loaded, location X'7000' will contain an X'CD', X'7001' will contain X'47' etc.

Yanked Patch Block - Type X'10'

This type of block is used exclusively by the LDOS PATCH utility. When an X-patch is installed in a load module file, a header block and series of object blocks (X'01's) are appended to the end of the file (overwriting the last Transfer address block). When the patch is YANKed, all of the object blocks belonging to that patch are changed to X'10's so that the system loader won't load them.

Transfer address Block - Types X'02' and X'03'

These types inform the loader where to begin execution once the entire module has been loaded into memory. Since it only takes two bytes to store an address, any bytes following the address with a length other than X'02' would be unused. In the following example:

```
        02 02 6A 3F
```

The transfer address of the module would be X'3F6A' (the address is stored in lsb,msb format). Most assemblers allow the transfer address to be specified in an END statement. The only difference between the X'02' and X'03' type bytes is that the latter indicates a file that is not executable.

Header & Comment Blocks - Types X'05', X'07', & X'1F'

These types do not actually load into memory at all. Most assemblers write an X'05' type as the first block of the file. It usually has a length of 6 bytes and the data following is most likely the first 6 characters of filename. The DOS PATCH utility generates the X'07' type block preceding X-patch data blocks. The data contained therein is the name of the patch file. This is necessary in order to YANK the patch by that name. Assemblers most likely do not generate this type of block. This and type X'10' are used primarily by the PATCH utility. It is sometimes desirable to insert comments into a program but not have the comments resident in RAM. A block type of X'1F' indicates a comment block to the loader. Note: none of these block types specified (X'05', X'07', X'1F') have any effect on the object code.

When FED2 recognizes a file as Load Module format, it reads through the file and maps every block. Once FED2 has successfully mapped the file, additional information relevant to that file is now accessible. Wherever the cursors are positioned in the file, the appropriate load module information is displayed. The following are the block types and the messages that will be displayed:

```
X'01' - Block X'nnnn' - X'nnnn'
X'02' - Transfer Address X'nnnn'
X'03' - Transfer Address X'nnnn'
X'05' - File Header Block
X'07' - Patch Header Block
X'10' - Block X'nnnn' - X'nnnn'
X'1F' - Comment Block
```

When positioned at actual object code in a load block, the following will be displayed:

```
        Load Address X'nnnn'
```

This is the memory location that will receive the byte at the current cursor position. An asterisk before this indicates that this byte would load at that address, except that this block has been yanked (see type X'10'). Alongside the load address will also be the mnemonic Z80 instruction. The <L> function of the <F>ind mode allows positioning the cursors to a byte in a file which loads into memory at the specified address. The <I> command positions the cursors to the next instruction. The <J> command locates the position that the current instruction refers to (either direct or indirect). Both of these commands allow for instructions spanning load blocks or sectors. With these commands, it is possible to step through a machine language program.

When in a load module file, some information will always be displayed, regardless of the current function mode (hex modify, ASCII modify, normal). When using any modify mode in a load module file, pay VERY close attention to what is being changed. If loader codes are ever overwritten accidentally, disastrous results may occur. This might be realized the next time the file is loaded. If the message "Load file format error" appears, this means that an illegal type byte was encountered. This could have happened because the type byte itself was changed, or a length byte was changed indicating the incorrect position of the next type byte.

In some cases, it might be desirable to view a load module file as a data file. This is accomplished by typing an "*" before the filespec at the prompt. This action will prevent the file from being mapped. Any string searching will encompass the entire file. including load blocks. For example, to hide a block of object code by changing an object block, type byte (X'01') to a yanked patch block (X'10') or comment block (X'1F'), or any other type byte. It is also possible to change the load address in an object block, of data in a comment block or patch header block. These actions are not common, but this feature is provided for advanced users.

Disk Structure

In order to store information of any type on a disk, it must be formatted. Formatting is the process in which the disk media's magnetic surface is organized into concentric circular regions called tracks. Five inch floppy disks can be formatted anywhere from 2 to 80 tracks. Typical standards are 35, 40 and 80, depending on the drive. Some disk drives have more than one head, in which case the term "cylinder" comes into play. A cylinder is a collection of tracks grouped together as one logical unit. For example, a double headed 40 track drive has 80 total tracks, but only 40 cylinders. Each pair of adjacent tracks on opposite sides of the diskette form a cylinder. Cylinders are numbered sequentially starting at the outside edge of the diskette, which is cylinder zero. Each track is divided into smaller units called sectors. The quantity of sectors per track depends on the disk type and density. Single density 5" floppy disks have 10 sectors per track, and double density 5" floppy disks have 18 sectors per track. Some hard drives have up to 32 sectors per track and 256 sectors per cylinder. Each sector contains smaller units known as bytes. One character of data is equivalent to a byte.

Disk Files

After formatting and verifying the cylinders on the disk, system information must also be written to that disk. "System Information" is a collective name for two disk files - BOOT/SYS and DIR/SYS. A disk file is a collection of sectors on a disk in a specific order. A file is referred to by means of its filename, extension, and drive number. This can be abbreviated to the term "File Specification" or FILESPEC. The first portion of cylinder zero is allocated to a file called BOOT/SYS, which contains information necessary for that diskette to boot up. One full cylinder is devoted to a file called DIR/SYS, also known as the directory. The directory is a collection of tables and maps used to determine anything about that disk - the disk name, password, and date of creation (specified during format), how much space is available, how much is used, what files exist, how much space they occupy and where those files are located on that disk. Any data stored on that disk is stored in a file. Each file is made up of logical units called records, referred to by a number in the range of 0 through 65535. Each record has a fixed length between 1 and 256 bytes, the most common being 256 (the same size as a physical sector). For example, the file named BOOT/SYS contains 5 records, each with a logical record length of 256. The information such as file length, record length, dates and status are collectively called attributes of a file. To see the attributes of any file, use the DIR command with the (A) parameter.

IFC

General information

Anytime you have more than one disk drive on-line, you develop a collection of disks containing many files. File maintenance of moving files among disks while purging unneeded files can become a clumsy series of DIR commands followed by COPY and or REMOVE commands. Using PURGE with its file-by-file query can sometimes be useless when you have forgotten the contents of the files - you need to list them. IFC gives you the ability to perform these maintenance tasks interactively. A menu-controlled screen provides the tools to easily list, copy, delete, or rename a file or groups of files. You will find IFC essential to the task of file maintenance.

Invoking IFC

IFC is easily invoked via the command syntax:

```
IFC [:d] [(X,Inv)]
```

d	Optionally specifies the logical drive number to work with.
X	Optionally indicates that drive 0 does not contain a SYSTEM disk.
Inv	Optionally specifies that you want INVISIBLE files included in the directory list
Abbr:	I=Inv

If no drive number was entered on the command line, IFC will prompt you for the working drive with:

```
        Select drive ( 0 – 7 ) ?
```

Enter the number of the drive you wish to work with. After the disk directory has been scanned and its contents sorted, the following information will be displayed:

```
FILENAME/EXT:D   *IP+  DD-MMM-YY HH:MM  xxxxK
```

The first column is the file specification. The second column contains the file's attributes. The asterisk [*] indicates that a file is a partitioned data set (PaDS). The uppercase "I" and "P" stand for invisible and protected file respectively. The plus sign [+] indicates that the file has been modified since the last time it was backed up. The third column contains the date and time (where applicable) on which the file was last modified. The last column is the file size rounded to the nearest K. This is not the amount of space that the file takes up on the disk but rather the size of the file. If the file had been tagged, then the filename would be preceded by an asterisk to indicate its tagged state.

The first file on the screen will have the character string "==>" pointing to it. We will call this string an "arrow". The file pointed to by the arrow will be referred to as the "current file". Any command which acts upon a single file will act on the file pointed to by this arrow. The copy, delete, tag and untag commands all affect the current file. To move this arrow, simply use the <DOWN ARROW> to advance to the next file in the list, or the <UP ARROW> to go to the previous file in the list. You may also depress the <SPACE BAR> to advance the pointer to the next file.

A <LEFT ARROW> or <SHIFT UP ARROW> will position the arrow to the top of the list whereas a <RIGHT ARROW> or <SHIFT DOWN ARROW> will position the arrow to the bottom of the list. You may also use <;> to advance to the next page of files or <-> to decrement to the previous page of files.

When the pointer advances to the start or the end of the list it will "wraparound" to the end or the beginning. IFC will also display a blank line to indicate that wrap-around has occurred.

Depress the <H> key and lFC will display the help menu. The help menu looks like this:

```
    A - Again, retag files      C - Copy current file
    0 - <Un>Tag old/new files   D - Delete current file
    T - Tag current file        L - List current file
    U - Untag current file      R - Rename current file
    W - Wildcard <un>tag        # - Execute program
!+I*P - <Un>Tag by attributes   E - Exit to DOS

== <M>ass functions ==          F - Free space on drive
    C - Copy files              H - Display help
    D - Delete files            Q - Execute DOS command
    M - Move files              S - Select new drive
    R - Rename files            ( - Alter parameters

             Press any key to continue
```

Any of IFC's commands may be aborted by depressing the <BREAK> key.

Current file commands

The following group of commands all deal with a single file. All of these commands will act on the current filespec.

<C>opy a file

This command will copy the current file to a specified drive. Depressing the <C> key will generate the prompt, *"Copy file(s) to drive ?"* Once you select the destination drive, the prompt, *"Reset MOD flags (Y/N) ?"* will be displayed. Depress <Y> and lFC will reset the modification flag in the directory of the source disk. Once the copy has been completed IFC will advance the arrow pointer to the next file.

<D>elete a file

Depress the <D> key and lFC will provide you with an opportunity to escape via the prompt, *"OK to delete file : filespec/ext:d ?"* Depress the <Y> key to delete the current file. Depress the <N> key or <BREAK> to abort. Once the file has been deleted, lFC will advance the arrow to the next file.

<L>ist a file

Depress the 'L' key and IFC will prompt, *"List file in ASCII (Y/N) ?"* Depress the <N> key to list the file in hex, or <Y> to list the file in ASCII. The file can be printed if the PRINT parameter is turned ON. If PRINT is OFF, the file will be listed to the screen. A screen listing will pause after each page of the file is displayed. Any key continues the listing.

<R>ename a file

Press the <R> key and IFC will prompt, *"New name for FILESPEC/EXT:D ?"* Type in the new filename for that file and depress the <ENTER> key. The file will be renamed. The arrow pointer will be returned to the start of the list.

<#> **Execute program**

If the arrow is currently pointed to a BASIC program file (file extension "BAS"), an executable program file (file extension "CMD"), or a Job Control Language file (file extension "JCL"), you can automatically terminate IFC and invoke that "file" with the <#> command. BASIC programs will be invoked via the command, **`BASIC current-file`**; JCL files will be invoked via the command, **`DO current-file`**; and executable programs will be invoked via the command, **`current-file`**. Note that IFC will terminate prior to execution of the program.

If the current file has no extension, or has one other than the three identified above, the <#> command will be ignored.

Tagging commands

When an IFC command is given that will work on a group of files, the files must be tagged.

<T>ag files

Depress the <T> key and IFC will tag the current file. An asterisk [*] will be displayed next to the file to indicate that the file has been tagged. IFC will also display a running total (in K) of all tagged files.

<U>ntag files

Depress the <U> key and IFC will untag the current file. The untag command works just like the tag file except in reverse.

<A>gain, retag files

Depress the <A> key and IFC will tag all of the files that were tagged and later mass copied. These files will be marked with a number sign [#] to indicate that the file was tagged and then copied.

<O>ld tag

This tagging command references the working disk against a target disk. Depress the <O> key and IFC will prompt you with, *"<O>ld or <N>ew files ?"* Depress the <O> key and all of the files which exist on both disks will be selected. If you depress the <N> key, all of the files which do not exist on the target disk but are in the currently selected disk will be selected. You will then be prompted with, *"<T>ag or <U>ntag files ?"* Depress the <T> key and all of the selected files will be set up for tagging, or depress the <U> key and the selected files will be set for untagging. Finally, you will then be prompted for the target drive with, *"Drive to scan (0 - 7) ?"* Depress the number of the drive to scan for the target drive. Your selection will then be invoked.

<W>ildcard tag

The wildcard tag command is used to tag or untag a group of files that match up with a wildcard file specification. Depress the <W> key and IFC will prompt with, *"<T>ag or <U>ntag files ?"* Depress <T> to have all matching files tagged, or <U> to have all matching files untagged. IFC will then prompt, *"Filename wildcard ?"* Enter a wildcard and IFC will either tag or untag all matching files. The wildcard file name and extension fields may consist of standard filespec characters [A-Z,0-9], question marks which match all characters in their respective position, and an asterisk which will force a match of all characters to the end of the field.

<!+IP*> Tag by attribute

This command allows you to tag a multiple of files which match the criteria associated by the command. The <!> command will flip the state of the files: tagged files become untagged, and untagged files become tagged.

The <+> modify tag command will tag all files which have the modified flag set in the directory. These files will be marked with a '+' on the display attribute field.

Similarly, the <I> invisible tag command tags all files which are classified as invisible to directory operations so indicated by the "I" attribute.

The <P> tag command will tag those files identified as protected by password protection or limited access capabilities.

Finally, the <*> command will tag those files which have the PDS bit set in the directory. Such files are usually Partitioned Data Sets supported by our PaDS utility or diskDISK host files supported by our diskDISK utility.

Mass commands

The Mass commands are used to act on all of the tagged or all of the untagged files. Depressing the <M> key will begin the selection of a "mass" command. IFC will prompt, *"Mass <C>opy, <D>elete, <M>ove, or <R>ename ?"* Make the selection by pressing the first letter of the function you wish to perform.

<M>ass copy

The mass copy command works just like a multiple copy command. IFC will copy all tagged files or all untagged files automatically. The selection is made by responding to the prompt, *"Copy tagged or untagged (T/U) ?"* Depress the <T> key to select all of the tagged files, or the <U> key to select all of the untagged files. Don't forget that <BREAK> will abort the operation. IFC will then prompt you for the target drive with, *"Copy file(s) to drive?"* Enter the destination drive for the files you wish to copy or depress <BREAK> to abort. IFC then prompts, *"Reset MOD flags (YIN) ?"* Answer <Y> and IFC will reset the modification flags in the source directory. IFC automatically resets the modification flags in the destination directory, but you may not want them cleared from the source directory. Depressing <BREAK> will abort the mass copy function.

<M>ass delete

For mass delete, IFC prompts, *"Delete tagged or untagged files (T/U) ?"* Depress the <T> key and IFC will purge all tagged files from the disk. If you depress the <U> key, IFC will purge all of the untagged files. Depressing <BREAK> will abort the mass file delete operation.

<M>ass move

The mass move command works just like the mass copy command except that it will delete the file from the logged drive after each file is successfully copied to the destination disk. IFC will move all tagged files or all untagged files automatically. The selection is made by responding to the prompt, *"Move tagged or untagged (T/U) ? "* Depress the <T> key to select all of the tagged files, or the <U> key to select all of the untagged files. Don't forget that <BREAK> will abort the operation. IFC will then prompt you for the target drive with, *"Move file(s) to drive?"* Enter the destination drive for the files you wish to move or depress <BREAK> to abort. IFC automatically resets the modification flags in the destination directory but you may not want them cleared from the source directory. Depressing <BREAK> will abort the mass copy function.

<M>ass rename

The mass rename operation lets you change the file name and/or extension for all of the tagged files. The new filespec is derived by passing the current filespec through a template. The template is entered in response to the query, *"Rename template ?"* and is identical in syntax to the wildcard identified above. Alphanumeric template characters will be passed to the new filespec. The question mark [?] will cause the correspondingly positioned character of the current filespec to be part of the new filespec. If the template contains an asterisk [*], the remaining part of the field of the current filespec: will be transferred to the new filespec.

Miscellaneous commands

<E>xit to DOS

Depress the <E> key and IFC will return to DOS Ready. Note that IFC also accepts the <X> key to specify exit.

<F>ree space

Depress <F> and IFC will prompt, "Free space on drive ?" Depress the number of the drive you wish IFC to scan. IFC will display the following information:

```
       Drive : D  Disk name : NNNNNNNN  Free space : xxxxxK
```

where "D" is the drive number, "NNNNNNNN" is the disk's name, and "xxxxx" is the total amount of free space on that diskette. Depress <ENTER> to continue.

<H>elp

Depress the <H> key and IFC will display the list of all IFC commands. Press any key to continue. Note that the <?> key is also accepted to specify HELP.

<Q> Execute DOS command

Depress the <Q> key and IFC will prompt for the DOS command with the query, "Enter DOS command:" Enter any DOS command and IFC will execute the command. You should be careful not to perform any DOS command which will change the contents of HIGH$ as IFC will protect itself above HIGH$ when executing the DOS command. When the command has completed, IFC will resume after you respond to the prompt, "Press any key to return to IFC..."

<S>elect new drive

This command is used to change the working drive. Depress the <S> key and IFC will prompt, *"Select drive (0 - 7) ?"*. Depress the number of the drive you wish to log in. Depressing <BREAK> will return you to the command prompt. If, for some reason, IFC cannot log in the new drive, it will again display the Select drive prompt. Once IFC has returned to the Log drive prompt for the second time, a <BREAK> will return you to DOS Ready.

<(> **Alter parameters**

This command is used to alter selected parameters, such as whether the list command will display or print the file designated, whether full verification including CRC error checking will be performed, or whether the file list should include invisible files. When you specify the <(> command, IFC will prompt, *"Parameter <V>erify, <I>nvisible, or <P>rint ?".*

If you enter a <V>, IFC will toggle the state of VERIFY; the current state will always be displayed in the display header.

If you enter an <I>, IFC will toggle the state of invisible files for inclusion into the file list and then re-log the current disk. The state of accepting invisible files is indicated by displaying a letter "I" immediately following the drivespec in the display header.

If you enter a <P>, the current state of the PRINT flag will be inverted. This is visually observed in the display header. When PRINT is ON, the <L>ist command will print the selected file rather than displaying it on the video screen.

PRO-CESS

PRO-CESS is a powerful maintenance tool for "CMD" or "CIM" type load module files. It provides for file appending, mapping, sorting, packing, offsetting, library member and partitioned data set member extraction, as well as the specified deletion of any load module record. PRO-CESS can convert "CMD" files which contain various types of records to "CIM" files which are pure binary core-image constructs. It also provides the capability of converting "CIM" files to "CMD" files. PRO-CESS gives capabilities to load module maintenance never before possible. PRO-CESS does it all: rapidly, totally, and economically!

What is PRO-CESS?

The PRO-CESS tool is a powerful machine language program that has been designed to provide total maintenance of program load modules on a record basis. This means that it references the load module as a multi-record type, variable length record file - just like the operating system loader. By using the various commands identified in the PRO-CESS menu, you can completely reorganize the load structure of a given module or modules in order to make them more efficient in terms of loading speed and occupied disk space.

PRO-CESS also provides the capability of converting "CMD" type load modules to/from "CIM" core-image load modules. This is especially useful to generate program files in "binary" form for PROM burners. The "CIM" structure is identical to the "COM" structure used in other types of operating systems.

You get the capability of appending two or more "CMD" machine language load module files into one file. This is useful to concatenate two or more separately assembled OBJECT code files, concatenate two or more noncontiguous blocks of code, or also couple two or more programs together so they load together. You get control of the program's transfer address or ENTRY point.

"CMD" files can be copied from one SYSTEM diskette to another SYSTEM diskette on a single drive system provided both diskettes use the same operating system.

You get the capability of totally mapping every record in a load module. Determine the TYPE as well as the load address range of each load record. This load map is displayed in the MENU status. You can optionally request that a map listing be sent to a line printer.

You can selectively remove any record in the load module. Get rid of spacewasting headers that are not necessary. Remove dead space. In case you make a mistake, you can even un-remove a removed record before clearing the load-module's memory buffer.

By far, the most powerful function included in this toot is the reorganization capability of the PACK command. This powerful function converts any X-type patches to D-type patches [X-type patches are generated by the LS-DOS Version 6 PATCH utility]. It then sorts the buffer by load address to construct sequential load records and generates a load module file that uses maximum sized (256-byte) load records. This feature is quite useful for reorganizing large inefficiently generated load modules such as Tandy's COBOL package. The PACK function is also useful for reorganizing the out-of-sequence load modules generated by the LC compiler [the records are out-of-sequence due to the separation of program and data regions during the compilation process].

The PRO-CESS menu

When you execute the PRO-CESS maintenance tool, all of its functions are immediately available through single letter commands. These commands are displayed in the PRO-CESS MENU. A reasonable facsimile of the MENU follows.

```
    PRO-CESS 2.0 [copyright (C) 1983 Roy Soltoff]

       <C>lear the buffer region
       <D>OS Command request
       <E>xit to DOS
       <I>mage file load/write
       <L>oad a file into the buffer
       <M>ap the buffer records
       <O>ffset address from current load origin
       <P>ack the buffer records
       <R>emove a record from the buffer
       <S>ort the load records by address
       <U>n-remove a "removed" record
       <W>rite the buffer to a disk file

Buffer: Size 46802 Used 00002 Free 46800 Records 00000
Module: Origin FFFF End 0000 Entry 0000 Offset 0000
```

The menu of commands contains each command letter within angle brackets, "<>". For instance, the command to *"<L>oad a file into the buffer"* can be selected by depressing the "<L>" key on the keyboard. Each of the commands displayed in the MENU may be selected by depressing whatever key is contained in the angle brackets. Notice that the "<L>" command has a large blinking graphics block preceding it. This "cursor" is used for an alternate means of command selection. If you depress the <UP-ARROW> key, the block will move up to precede another command. The <DOWN-ARROW> key will move the block down to the next lower command. The block will wrap around in either direction. When PRO-CESS is waiting for you to enter a command request, this cursor block will blink. When a command is being processed, the cursor block will not be blinking. Any menu command that is preceded by the graphics block can be selected just by depressing the <ENTER> key in lieu of the command letter. This feature is provided for your convenience. When you use the command letter to select a command, the block will be automatically moved to that menu command so you will have a visual indicator as to the command currently in progress.

In order to provide maintenance on a load module file, the file must be loaded into the PRO-CESS memory buffer. The MENU will constantly display the status of this buffer via the status line titled, *"Buffer:"*. The status line contains four fields: Size, Used, Free, and Records. The "Size" field will display the total number of bytes provided for the buffer. This value is determined from the memory available between the end of the PRO-CESS program and the highest memory address available based on the "HIGH$" value. The "Used" field contains the number of buffer bytes that are currently in use. Each record that is loaded into the buffer requires a two-byte linkage pointer in addition to the memory taken up to store the record. Two bytes are initially used to store the "head" linkage pointer. The *"Free"* field shows you how many bytes are available for use. This field is the difference between *"Size"* and *"Used"*. The *"Records"* field maintains a count of the total number of records stored in the buffer.

The MENU also displays the status of the program load records currently stored in the buffer. This status covers the program's ORIGIN or lowest address, its END or highest address, its ENTRY or transfer execution address, and any OFFSET specified. The OFFSET is a maintenance function that can be used to construct a file which loads into an address space different from where it executes.

The line of dots represents a third status/prompt line which will display informational messages pertinent to PRO-CESS commands and prompting messages where required. It is also used to display any error message returned by the operating system during service requests.

Re-entering PRO-CESS after inadvertent exit

It sometimes happens that you <E>xit the program without <W>riting the buffer to a file. Rest assured that you can recover from this mishap. If you immediately execute:

```
PROCESS *
```

the program will not automatically <C>lear the buffer region. The asterisk, "*", provides the needed error recovery. Use the <M>ap command to scan through the buffer to ascertain its validity before attempting to generate a file.

Command details

The following sections will describe the function and operation of all PRO-CESS commands identified in the MENU.

<C>lear the buffer region

The <C>Iear command is simple enough - it restores the buffer as if you just executed the program. Since this action will automatically clear any load module contained in the buffer, PRO-CESS gives you a second chance to acknowledge your selection. The MENU status will display the prompt:

```
        Are you sure you want to clear the buffer <Y,N> ? >
```

By entering a response of <N>, you will abort the <C>lear selection. A <BREAK> will also abort the selection. Only by entering a <Y>, will the <C>lear function activate. When the buffer is cleared, you will observe a "flash" of the video display as the MENU is re-generated.

<D>OS Command request

The <D>OS Command function provides access to operating system commands from the MENU level. Your DOS requests should be limited to library commands [a summary of library commands is normally obtainable via the "LIB" DOS command]. You enter your command request in response to the prompt:

```
        Command? >
```

After your command is entered, the MENU will be erased while your command line will be displayed at the top of the video display screen. When the command that you requested is completed, you must depress the <ENTER> key to refresh the MENU display. This provides the opportunity of analyzing any screen image displayed by the DOS command before the MENU display erases the image.

<E>xit to DOS

This command provides the means to terminate the maintenance session and return to DOS.

<I>mage file load/write

This command provides two functions - both relating to core-image files. Use the <I>mage command to load a core-image file from disk into the buffer. You also will use the <I>mage command to write the buffer out as a core-image file. To help with your selection, the MENU will display the prompt

```
       Image file LOAD or WRITE <L,W> ? >
```

A response of <L> invokes the image loading function which subsequently requests you to enter the image file specification via the prompt:

```
       Input file specification >
```

If you omit the file extension, PRO-CESS will use "/CIM" as the default extension.

Since core-image files have no loading information contained in the file, it is necessary to specify the origin address of the module. You do this in response to the prompt:

```
       Enter the module origin or <ENTER> to use [0000] >
```

Your selection must be entered in hexadecimal. Also, as can be noted by the prompt, if you depress just the <ENTER> key without a value. a default origin of X'0000' will be used. This value will also be used for the ENTRY or transfer address. The entire file, including the full last sector, will be considered as part of the program.

If you respond to the initial prompt with a <W>, you will invoke the image writing function. It is essential that the load records stored in the buffer be in sequential load order. The <IW> function will first scan the load records to ensure that they are, for a core-image file cannot be constructed if the records are out of order. If a problem is detected, the MENU will display the error message:

```
       Buffer is not in sequential load order!
```

It is not necessary for the load records to be contiguous. The <IW> function will generate null bytes, X'00', for all addresses interstitial to two adjacent non-contiguous load records. You identify the file specification of the file to be written by responding to the MENU prompt

```
        Output file specification >
```

If you omit the file extension, the default value of "/CIM" will be used. Upon successful completion of the file generation, the message:

```
        Requested file now written
```

will be displayed and PRO-CESS will await your next MENU selection.

<L>oad a file into the buffer

The <L>oad function is used to read a load module file into the memory buffer. The file will be appended to any already contained in the buffer. You identify the specification of the file in response to the prompt:

```
        Input file specification >
```

If you omit the file extension, the default value of "/CMD" will be used. While the file is being read into memory, PRO-CESS analyzes it to determine the specific type of load module: OBJECT file, ISAM overlay file, or a partitioned data set (PDS) file [the PDS file structure has been defined by MISOSYS]. An object file will be loaded directly into the buffer.

If the file is recognized as an LS-DOS structured ISAM overlay file, you will need to identify the overlay number of the desired member. In general, system files SYS6/SYS, SYS7/SYS, and SYS8/SYS are ISAM overlay files. These numbers are listed later. You will be requested to enter this ISAM number entry with the prompt:

```
        File has ISAM overlays - enter # >
```

This number is entered in hexadecimal. If you respond to the prompt by depressing the <ENTER> key without a number, the entire file will be loaded into the memory buffer. The only purpose for doing this would be to map the file as no reorganization is possible with this maintenance tool.

If the file is recognized as a PDS file, you need to specify a MEMBER specification. This is done in response to the prompt:

```
        File is a partitioned data set,
        enter MEMBER >
```

The MEMBER is the eight-character member specification as observed from a directory display of the PDS members. If you respond to the prompt by depressing the <ENTER> key without a MEMBER, the PDS Front End Loader program will be loaded into the memory buffer. Since it is likely that any given PDS contains non-CMD members, the PRO-CESS maintenance tool does not attempt to read the entire PDS file into memory.

If you specified an ISAM number or MEMBER that can not be located in the file, the error message:

```
        Requested ISAM member is not in the file!
```

will be displayed. It is highly improbable to receive the error message:

```
        Overlay beyond end of file!
```

however, if you do, it means that the ISAM directory contains a location for the member that is not within the scope of the file. You probably have an error in the file.

If any disk I/O error results while the file is being read, or any problem occurs that results in the file not being read to completion, PRO-CESS will return to the MENU command request after displaying an appropriate error message. No fragment of the file will be added to the memory buffer.

If there is no more available space in the memory buffer during the loading of a file, the error message:

```
        Insufficient buffer space to load file!
```

will be displayed.

If you attempt to load in a file that is not a load-module structure file, PRO-CESS will display the error message:

```
        File is not a load module file structure!
```

and the load operation will cease.

Upon successful completion of the load operation, the message:

```
       File is now loaded into the buffer
```

will be displayed. The buffer and module status lines will be updated to reflect the revisions made to the buffer with the load of the file.

A <BREAK> detected during the loading of a file will immediately abort the loading operation. No fragment of the file will be added to the memory buffer.

<M>ap the buffer records

This MENU command provides the function of mapping the buffer contents. Mapping is useful for obtaining the record number of records you wish to remove. It is also helpful to understand how unorganized your load module file is. The prompt message:

```
       MAP output to printer <Y,N> ? >
```

gives you the option of directing a load-module map to a printer by responding with <Y>. If you respond with <N>, only the status line will display the mapping, one record at a time. If you do not select a printer map, it is necessary to depress the <ENTER> key to obtain the mapping information for each record.

Each record will be given a sequential logical record number. This number is used as a record reference in the <R>emove and <U>n-remove MENU commands. Records will be identified as to type: Module header, Yanked load block, Load, Transfer Address, and so forth. The address range of load records will also be displayed.

If you specified the printer option, the printer will first be checked for availability. If it is not ready for use, the message:

```
       Printer is not available!
```

will be displayed until it is made ready. The <BREAK> key can be used to escape from this condition.

<O>ffset address from current load origin

Offsetting a load module means changing its loading address so that it loads into memory at a location different from where it was assembled to execute. There are a few reasons for wanting to offset a file. One, of course, is to offset a file assembled to run from a PROM so that it loads into a RAM region usable by a PROM burner. The <O>ffset MENU command requests the revised load origin via the prompt:

```
        Enter the offset origin address >
```

This entry is to be made in hexadecimal. For example, if the existing load module origin is X'3000' and you want it to load starting at X'5300', enter the four-character value, <5300>.

<P>ack the buffer records

The <P>ack operation is the most powerful feature of PRO-CESS. It is used to reorganize an object load module file so that it is most efficient in disk storage space and optimum for rapid loading by the operating system. Packing is a three-phase operation. The first phase identifies any LS-DOS X-type patch records and packs the object code revisions into the preceding load records wherever possible. Any patch address that is outside the range of the existing load records, is used to generate new load records. The second phase then uses the <S>ort facility to sequence the load records by sequential load address. The third and final phase generates a new object load module file with maximum-sized load records. This is achieved by combining short contiguous load records wherever possible.

The first two phases require no action from you. Appropriate status messages are displayed to apprise you of the phase. The first phase is identified by the message:

```
        Packing any "X" patches ...
```

The second phase is noted by the sorting message as noted in the <S>ort command discussion. The third phase will generate the dialogue as noted in the <W>rite command discussion.

<R>emove a record from the buffer

This MENU command can be used to delete an entire record from the load module. You must identify the record by number. The record's number can be identified with the <M>ap command. The record is deleted by setting a "removed" flag for the record which is bit-7 of the record's TYPE byte. For instance, a load record will be changed from TYPE=01 to TYPE=81. If you map the buffer after removing a record, you will observe the change. Any record that is "removed" will not be written to a file during the <W>rite or <I>mage <W>rite commands.

<S>ort the load records by address

This command reorganizes the buffer's load records [record type 01] so that they are in sequential load order. If non-load records are intermixed between the load records, they may not maintain their position after the buffer is sorted.

If the buffer contains any X-type patch records that were generated by the LS-DOS utility, PATCH, the program may be adversely affected by sorting. To apprise you of this situation, the <S>ort command will display the message:

```
X-patches present.
Do you still want to sort <Y,N> ? >
```

If the patch type-01 records are totally outside of the load range of all nonpatch type-01 load records, you may proceed to sort the buffer. If no patch type-01 record extends into the range of a non-patch type-01 record, you also may proceed with the sort [this implies that a type-01 patch record is wholly contained within the range of a previous type-01 record]. If you are unsure of the consequences, do not sort. An alternative is to use the <P>ack command which packs any X-type patches into the non-patch area of the buffer space.

The sort operation will commence with the display of the message:

```
Buffer sort commencing ...
```

If the buffer contains a very large file, the sorting may take a half-minute or more. A blinking asterisk will amuse you while you await the completion of the sort. Upon completion, the status message:

```
       Buffer is now sorted
```

will be displayed. Note that the record numbers are changed if the sorting process detects any out-of-sequence load record.

<U>n-remove a "removed" record

If you inadvertently remove the wrong record with the <R>emove command, you can recover from your error with this <U>n-remove function. As previously stated, the record number is obtained from a buffer mapping operation.

<W>rite the buffer to a disk file

This command is used to write the buffer contents to a disk file. Since you may have appended two or more modules into the buffer, you now have the opportunity of changing the module's ENTRY address as noted in the MODULE's status display. This capability is useful when appending two or more files since the transfer address used would default to the transfer address of the last file loaded. Respond to the prompt:

```
       Enter new ENTRY address or
       <ENTER> to use [xxxx] >
```

If you want to change the transfer address (entry point), you can enter the new address in hexadecimal. If you want to maintain the ENTRY as specified in the MODULE's status line [and also repeated in the prompt], just depress the <ENTER> key.

You identify the file specification of the file to be written by responding to the MENU prompt:

```
       Output file specification >
```

If you omit the file extension, the default value of "/CMD" will be used. Upon successful completion of the file generation, the message:

```
       Requested file now written
```

will be displayed and PRO-CESS will await your next MENU selection.

Appending two or more files

You may have the occasion to assemble two separate object files that you want to combine into one. In order to append or concatenate two or more files into one contiguous file, use the <L>oad command to combine the files into the memory buffer. When the last file has been loaded, use the <W>rite command to generate the combined object load module file. Note that the transfer address of the concatenated file would be the transfer address detected from the last file input. The <W>rite command provides the opportunity of modifying the transfer address to one of your choosing.

Customizing an LS-DOS library with PaDS

Since PRO-CESS provides the capability of extracting ISAM members from LS-DOS libraries, and PaDS provides the capability of building USER libraries, we can combine the power of both utilities to customize a USER library with DOS library commands.

If you execute an LS-DOS LIB command, you will see LIB <A>, LIB <B>, and LIB <C> commands displayed. The names of each command represent the entries to members in SYS6, SYS7, and SYS8 respectively. The command interpreter which resides in SYS1 compares your command entry to a table which contains ISAM numbers for each LS-DOS LIBrary command. It is these numbers that are needed to extract one of the LIBrary members. Here is a list of the codes:

SYS6-LIBA	**SYS6-LIBA**	**SYS7-LIBB**	**SYS8-LIBC**
---------	---------	---------	---------
31-APPEND	41-LIST	51-ATTRIB	B1-FORMS
20-CAT	81-LOAD	11-AUTO	B2-SETCOM
24-CLS	1E-MEMORY	33-BUILD	B3-SETKI
32-COPY	18-REMOVE	13-CREATE	A2-SPOOL
61-DEVICE	53-RENAME	15-DATE	1C-SYSGEN
21-DIR	63-RESET	14-DEBUG	A1-SYSTEM
91-DO	64-ROUTE	71-DUMP	
66-FILTER	82-RUN	22-FREE	
26-ID	65-SET	72-PURGE	
19-LIB	25-TOF	16-TIME	
62-LINK		1B-VERIFY	

Optimizing an inefficient load module

You just purchased that new COBOL compiler and are perturbed at how long it takes to load - not to mention the disk space it takes up. When you load it into PRO-CESS and perform a <M>apping, you are amazed to find that "RUNCOBOL" and "RSCOBOL" are generated with 16-byte load records. Use the <P>ack command to reorganize the inefficient RUNCOBOL file and shrink it from 1537 records to 97 records while at the same time you reduce the amount of disk space taken up by the file from 121 sectors (31.5K) to 98 sectors (25.5K) - a savings of 6K of disk space.

Disk load module formats

A load module is simply a disk file that can be loaded into memory by the system loader. The file is made up of variable length records and is usually a program. Many different types of records are included in a load module the DOS makes extensive use of distinct record types in load modules. One record type is a load record which contains information on where it is to load into memory. If the file can be directly executed as a program, it then becomes known as an executable load module (ELM). The usual term that has been applied to such a file is "CMD". That's because a directly executable load module can be invoked as if it were a system CoMmanD. We commonly use the file extension of "/CMD" for these command files.

A load module can be conceptualized as a sequence of RECORDS. Note that we did not say an ordered sequence. Thus, the implication is that the records do not have to be in an ascending order (contiguous load addresses). Each record contains three fields: a TYPE field, a LENGTH field, and a DATA field. It has a one-byte indicator as to what TYPE of record it is. This TYPE code is used to denote a record as a HEADER record, a TRANSFER record, an ISAM directory entry record, a LOAD record, or other meaningful structure. Each record also has a one-byte LENGTH field which is the length of the data area field. The data field length thus ranges from <1-256> in value (a 0 implies 256). The remaining part of the record is its DATA AREA and is used to store program code, directory information, messages, or other pertinent information. If you are familiar with BASIC random access files, you will see the similarity in the fielding of records - except in this case, we have variable length sequentially accessed records [with partitioned data sets provided in the PaDS utility, you also have variable length indexed sequential accessed records]. Figure 1 lists the various TYPE codes currently used in operating systems.

Figure 1: Load Module TYPE Codes

TYPE	DATA AREA
01	Object code load block
02	Transfer address
03	End of load-only program
04	End of partitioned data set member
05	Load module header
06	Partitioned data set header
07	Patch name header
08	ISAM directory entry
0A	End of ISAM directory
0C	PDS directory entry
0E	End of PDS directory
10	Yanked load block
1F	Copyright block

Any code above X'1F' is invalid as a record type. In addition, any code not listed in figure 1 is reserved for future use.

If you could look at a sample object program file, you would notice that it starts out with something like:

```
05 06 50 52 4F 43 45 53 1F 1E 43 6F ...
 .  .  P  R  0  C  E  S  .  .  C  o ...
```

stretched across the screen. What you have here is a load module header (TYPE=05). The length byte (LENGTH=06) follows the TYPE code. The 6-byte DATA AREA field is the header name. All records follow this "fielding" order. A record is organized with a TYPE, LENGTH, DATA sequence. The X'1F' begins the second record. It happens to be a copyright record with a LENGTH of VIE' or 30 decimal bytes. Incidentally, the TYPE=1F record is generated automatically by the "COM" pseudo-op in PRO-CREATE, the macro-assembler used to develop and maintain the LS-DOS operating system.

Note that each record begins with the TYPE code and the first byte following the end of a record is always the TYPE code of the next record. The only exception is when a TYPE code indicates the end of a file. If you look further in the record displayed at relative position X'28', or if you count 30 bytes down from the "C" of "Copyright", you will see:

```
01 02 00 30 ...
```

The record TYPE is a load block (TYPE=01). and the length of the data area is X'02', or 258 data bytes. Yes, we previously stated that the length ranged up to 256 and here we have 258! This TYPE-01 record is a special case. The two-byte field following the LENGTH is the starting load address for the rest of the field. Since the LENGTH value includes the 2-byte load address, a length of X'03' would indicate only one load byte. A length of X'04' would indicate two load bytes. A length of X'FF' would indicate 253 load bytes. A length of X'00' would indicate 254 load bytes. To be able to have a data area with up to 256 bytes of loadable data, the LENGTH values of X'01' and X'02' are indicative of 255 and 256 load bytes respectively. This is accomplished by having the system loader decrement the length value by two when reading a load address. The resultant value becomes the true length of the loadable data.

If you could look at the last four bytes of the file, they appear as:

```
02 02 00 30
```

This will represent the TRANSFER record (TYPE=02). Again, we have a LENGTH byte which shows a 2-byte data field. The data field contains the transfer address or entry point to the program in standard low-order, high-order sequence. The system uses this address as an entry to the program after successfully loading it into memory. This address is also what is returned in register pair HL by the @LOAD SuperVisor Call.

So far we have discussed the HEADER, the COPYRIGHT, the LOAD, and the TRANSFER records. These are the four common record types you will find in most load module files. We also observe that our discussion of program load modules was limited to a single program per file. Another kind of file is one that contains many program modules (or data modules) as sub-files. Since the file is divided into sub-files, it is considered a "partitioned data set" abbreviated as "PDS". The PDS contains a directory of its sub-files with each sub-file being termed a MEMBER of the PDS and having an entry in the directory. The system loader supports a particular kind of PDS used to contain the library overlays used in LS-DOS.

If you could look at a library file, you would see something like:

```
08 06 21 00 24 00 00 CB 08 06 61 ...
```

The TYPE code of X'08' indicates an ISAM DIRECTORY ENTRY record. The LENGTH byte denotes a DATA area of six bytes. After the sixth byte, you will see another TYPE=08 starting another ISAM directory entry record. The file is a partitioned data set. The TYPE=08 records are the directory entries for its members.

The ISAM directory data area is used by the SYSTEM loader to locate where a particular member can be found in the file. The data area includes positioning information indicating the exact byte position in the PDS which is the first record of the member. The six-byte data field is further divided into sub fields. The first byte (in this case, X'21') is the ISAM entry number. This entry number is provided to the system loader when a library command is parsed by the command interpreter. The entry number is the PDS member that will execute your request. The system loader searches the PDS directory for a matching directory record. The next two-byte sub-field is the transfer address of the member. The transfer address is contained in the directory so that more than one transfer address can be applied to a member. Therefore, a member can have multiple entry points. The last three-byte field is the triad pointer which points to the first byte of the member. The triad pointer is composed of the Next Record Number (NRN) and Relative Byte Offset for the member's first record byte. The system then positions to the pointer and loads the member. Thus you have six bytes of data as specified by the LENGTH byte. Since the process uses an index (the directory) to locate the member's starting byte then proceeds to sequentially read the member, the access method is termed "Indexed Sequential Access Method" (ISAM).

A TYPE-08 record can also have a 9-byte data area. In the PaDS utility available from MISOSYS, the ISAM directory entry record includes a three-byte subfield which contains the TRUE length of the member. The position of a member's logical end-of-file (EOF) can thus be calculated by adding its length to its position and adjusting for sector boundary alignment.

If you could look at the first byte following the last TYPE-08 record, you would observe the sequence:

```
0A 01 00 04 01 00 01 02 00 26 ...
```

The TYPE=0A indicates that it is the end of a PDS directory. The SYSTEM loader will return a "file not found" error if it reaches this record without finding a match of the ISAM number. The LENGTH=01 is needed because ALL load module records MUST have a length byte. The DATA area contains only a single arbitrary byte, X'00'. We cannot indicate a null record because a length byte of X'00' indicates 256 data area bytes. Thus, the X'0A' record type must have a minimum of one byte in its data area.

The following record is a TYPE=04 to indicate the end of a PDS member. This record serves but one purpose when used immediately following the directory - it will result in the return of a "Load file format error" if a library file is executed as if was a CMD file. When not expecting a partitioned data set file, the SYSTEM loader will ignore record types other than X'01' and X'02' except for the X'04'. The file reading will terminate at the X'04' with the above-mentioned error message.

The record type X'04' is usually used at the end of each partitioned data set member. Each member will usually end with "04 01 00" rather than a TYPE=02 record. The system loader uses the X'04' type code in lieu of the transfer address code because the SYSTEM loader recovers the transfer address from the ISAM directory. Thus it needs to take action different from that when a standard load file has been completely loaded.

The next record types to discuss are those used in a generalized PDS file as exemplified in the PaDS utility. Such a file starts with a record type X'06' in lieu of an X'05' which is the normal header type for a load module. This is used in certain utility commands to note whether the referenced file is a partitioned data set compatible with PaDS utilities.

The partitioned data sets include a *MEMBER DIRECTORY* which correlates the member NAME with its associated ISAM entry number. A representative *PDS MEMBER DIRECTORY* entry looks like this:

```
0C 0B 64 69 72 20 20 20 20 20 01 01 7A 0C ...
 .  .  d  i  r                 .  .  z  . ...
```

The TYPE=0C record indicates a PDS member directory entry record. The LENGTH byte specifies that the data area is an 11-byte field. The DATA AREA is subfielded as an 8-byte member name (stored in lower case), a one-byte ISAM entry number that is used to match up with a corresponding ISAM directory entry record, and a 2-byte field of member data. The first byte uses bit-7 to indicate a data member in contrast to an executable CMD program. Bit-6 indicates that the member has been established as "sector-origin". Bit positions 54 store the two high-order bits of the 5-bit year. Bits 3-0 and the next byte contain the 12-bit DATE field formatted as in the standard directory entry record. This entry is the modification date of the member which was added to the PDS. The end of the MEMBER DIRECTORY is indicated by a TYPE=0E record with its expected length and data field (as in "0E 0100"). The purpose of this record is similar to the TYPE=0A record for the ISAM directory. It indicates the end of the MEMBER directory. The ISAM directory is positioned in the PDS to follow the MEMBER directory.

One last set of record types to discuss is the records associated with the LS-DOS PATCH utility. When you apply an X-patch to a file, the name of the patch file is used as a header name with a record type of X'07'. Thus. if you want to YANK the patch, the PATCH program can read through the file and search for a like-named header. If a matching header is found, PATCH will change the header record type to a X'09' to indicate a yanked patch. Also, since it may be impossible to remove the patch without bubbling up any code blocks following the patch (another patch maybe?), PATCH will change the TYPE=01 records to TYPE=10 records. The TYPE=10 records will not be loaded by the SYSTEM loader but will be considered as non-loadable comment records.

ZCAT

ZCAT creates and maintains a catalog of all files which reside on your LS-DOS 6.3 (or TRSDOS 6.x) formatted diskettes. Each catalog file can store the directory file information on up to 255 diskettes or disks. Approximately 2000 file specifications are supported per catalog file. This number varies with the amount of memory available.

ZCAT is the professional way to search for the diskette containing a desired file. Because the diskette name (pack ID) is so important to isolate specific diskettes, it is important for you to maintain distinct names on your disks when formatting them. ZCAT or your DOS "ATTRIB" command can be used to rename a diskette where that function becomes necessary.

Invoking ZCAT

The ZCAT utility allows you to create and maintain a catalog of all files which reside on any LS-DOS 6.3 or TRSDOS 6.x compatible formatted diskette. It is invoked via the syntax:

`ZCAT (Inv,Sys,Page=nn)`	
`Inv`	Allows the cataloging of invisible files. Defaults to OFF.
`Sys`	Allows the cataloging of system files. Defaults to OFF.
`Page=nn`	Sets the printed page length (default=66)
Abbrev:	I=Inv, S=Sys, P=Page

When ZCAT is typed from DOS Ready, the program will load and display the initial logo and version number. After the initialization has been completed, you will be prompted with the "GETSPEC Menu" as follows:

```
Enter drive ( 0 - 7 ) or <BREAK> to exit
```

Selecting a number <0 - 7> will display all files on that drive with a /CAT extension. "CAT' is the default extension of directory catalog files for use with ZCAT. Pressing <ENTER> will bypass this prompt. Pressing <BREAK> will return you to DOS Ready.

You will then be prompted:

```
        Catalog filespec ? ..............
```

Enter the name of the catalog file you wish to read or create. The file specification may be composed of a file name of up to eight characters in length with an optional drive specification. The extension of "/CAT" will be added to the file specification automatically by ZCAT.

After the file specification has been entered, the master menu will be displayed and ZCAT will read in the catalog file if it already exists. Because the maximum number of files which ZCAT can hold changes with the amount of free memory available, it is possible to get the error message:

```
        * * File too large for available memory * *
```

If this occurs, press <ENTER> to return to the master menu and <E>xit to DOS. Reduce the amount of high memory which is allocated to DOS and again run ZCAT.

Once the catalog file has been read, the MASTER MENU will be displayed.

```
*> M A S T E R M E N U <*

    <A>dd disk to list
    <U>pdate disk in list
    <C>hange a disk name
    <R>emove a disk from list
    <S>earch for a file
    <D>isplay files on a disk
    <L>ist disks on file
    <P>rint files in list
    <E>xit to GETSPEC

        Selection ?

CAT file : MARC/CAT:0   Files cataloged :  324
Disks cataloged :  15   Maximum # files : 2226
```

The desired function may be selected by pressing the letter of the function which is bracketed between the "<>" symbols. In this display, the *"DIR file"* field will display the current catalog file specification [MARC/CAT is shown for illustration], the *"Disks cataloged"* field will display the quantity of disks cataloged in the catalog file, the *"Files cataloged"* field will contain the total number of file specifications cataloged, while the *"Maximum # files"* field will display the upper limit based on the memory currently available. The following sections will explain the use of each of the functions.

<A>dd disk to list

The <A>dd function will scan a disk that has not been previously cataloged and add the disk to the catalog list. All non-system visible files that reside on the disk will be cataloged [invisible and or system files may be cataloged if those options are selected at the invocation of ZCAT]. Press the <A> key from the master menu and you will be prompted:

```
        Scan which drive ( 0 - 7 ) ?
```

Enter the number of the drive which contains the disk you wish to be scanned, or press <ENTER> to scan the last accessed drive. The identification of the last accessed disk drive will be displayed at the bottom of the screen. After the drive has been selected, ZCAT will scan the directory and add all of the non-system, visible files to the list [see ZCAT's INV and SYS command line parameters]. If the disk has already been cataloged the error message:

```
        * * Disk is ALREADY cataloged
```

will be displayed. You may press <ENTER> to return to the master menu. Remember, each disk cataloged must have a name that is unique to the disk.

After the disk has been cataloged, ZCAT will sort the directory list and will again prompt you for the drive to scan. This gives you an easy way to catalog a number of diskettes via one keystroke. To return to the master menu, press <BREAK> at the prompt. This function only changes the working copy of the catalog in memory. The actual changes to the catalog file are made via the "<E>xit to GETSPEC" command, under your control. This is a safeguard for your protection.

<U>pdate disk

The <U>pdate function will scan the directory of an ALREADY cataloged disk. It will update the directory file to reflect any changes in the free space on the disk or any changes in the contents of the disk. Press the <U> key from the master menu and you will be prompted:

```
        Scan which drive ( 0 - 7 ) ?
```

Enter the number of the drive which contains the disk you wish to be scanned, or press <ENTER> to scan the last accessed drive. The drive last accessed will be displayed at the bottom of the screen. After the drive has been selected, ZCAT will scan the directory and update the directory list to reflect any changes that have been made since the disk was last <U>pdated or <A>dded. If the disk has NOT already been cataloged, the error message:

```
        * * Disk is NOT cataloged
```

will be displayed. You may then press <ENTER> to return to the master menu. Remember that a disk must be <A>dded before it can be <U>pdated.

After the disk has been scanned, ZCAT will sort the directory list [in case files have been added to or deleted from the diskette] and will again prompt you for the drive to scan. This provides you with an easy method to update the catalog for a quantity of diskettes via one keystroke. To return to the master menu press <BREAK> at the prompt. This function only changes the working copy of the catalog in memory. The actual changes to the catalog file are made via the "<E>xit to GETSPEC" command, under your control. This is a safeguard for your protection.

<C>hange a disk name

The <C>hange function will allow you to change a diskette's name from within ZCAT. Press <C> from the master menu and ZCAT will prompt:

```
        Which drive contains disk ( 0 - 7 ) ?
```

Pressing <BREAK> will return you to the master menu. After the drive has been selected, the current disk name will be displayed. You will be prompted for the NEW disk name. Enter the new disk name or depress <BREAK> to return to the master menu without changing the disk name.

After the name has been changed, ZCAT will wait for <ENTER> to be depressed before returning to the master menu.

<R>emove a disk from list

The <R>emove function will delete all traces of the disk from the catalog's directory list. Press <R> from the master menu and ZCAT will display the names of all disks which are currently in the directory list. ZCAT will pause after displaying a screen full of disk names. Press <ENTER> to continue the display or press <BREAK> to be prompted for "Disk name ?". Enter the name of the disk you wish to remove from the list or press <BREAK> to return to the master menu. If the disk name cannot be found you will be prompted:

```
* * Disk name NOT found
Press <ENTER> to return to the master menu.
```

After the diskette has been removed from the catalog's directory list, the list will be sorted and you will be prompted to press <ENTER> to return to the master menu. This function only changes the working copy of the catalog in memory. The actual changes to the catalog file are made via the *"<E>xit to GETSPEC"* command, under your control. This is a safeguard for your protection.

<S>earch for a file

The <S>earch function will allow you to rapidly locate a file or a group of files in the directory list. Pressing <S> from the master menu will prompt the question:

```
Search string ? ............
```

Enter the search string or press <BREAK> to return to the master menu. The search string may be a filename, a partial filename, or an extension. The search string may also contain a wild card character ("$") which may be used to mark a position as "don't care".

The partial filespec will display all files that begin with those characters. For example, a search string of "LB" would return any filenames which have the first two characters of "LB".

The extension will display all files which have the same extension. A search string of "/CMD" would display any files which have an extension of "/CMD".

The dollar sign ("$") wild-card character, may be used to mark a character position as "don't care". For example, a search string of "F$R" would display all files in which the first character is an "F" and the third character is an "R". Note that the second position can be any character.

The following are some examples of possible search strings and possible matches:

Search string	**Possible matches**
B/CMD	BASIC/CMD, BPT/CMD
$A/CMD	BACKUP/CMD, BASIC/CMD
/$$T	MOD4/DCT, TEST/TXT

ZCAT will then display all files in the directory list which match the search string. The filename will be followed by an asterisk ("*") if the file is a Partitioned Data Set. Along with the filename will be displayed the modification date of the file and the disk on which the file is located. ZCAT will pause after displaying a screen full of files. Press <ENTER> to continue the display or press <BREAK> to return to the master menu.

<D>isplay files on a disk

The <D>isplay-files-on-a-disk function will display a list of all files which reside on a particular disk. Press <D> from the master menu and all of the disk names cataloged will be displayed. ZCAT will pause after displaying a screen full of disk names. Press <ENTER> to continue displaying disk names or press <BREAK> to be prompted for *"Disk name ?"*. At the disk name prompt enter the name of the disk whose files you wish to view, or press <BREAK> to return to the master menu. If the disk name cannot be located you will be prompted with:

```
        * * Disk name NOT found * *
```

You may press <ENTER> to return to the master menu.

ZCAT – Disk Cataloger

If the disk name has been found, ZCAT will display the disk name and the free space available on the disk at the top of the screen. ZCAT will then display an alphabetical list of all files on the disk. ZCAT will pause after displaying a screen full of files. Press <ENTER> to continue the display, or press <BREAK> to return to the master menu. The following information will be displayed for each file:

Filename	Followed by an "*" if the file is a Partitioned Data Set
Protection	Access level; i.e. full, read, exec, etc.
LRL	Logical Record Length; 1 to 256
# of Records	number of logical records
Size	the amount of space that the file takes up on the disk, rounded to the nearest K (1K = 1024 bytes)
Mod date	the date the file was last written to
Disk name	the disk name where the file is located

A sample of this listing follows:

```
Disk name : MARC0037                                 Free Space :     0K
Filespec      Prot   LRL   #Recs   Size     Mod  Date   Disk name
================================================================
ATOD/ASM      Full   256      2      1K     27-Sep-82    MARC0037
CASSCO/ASM    Full   256     34      9K     18-Jun-82    MARC0037
CASSCO/CMD    Full   256      4      1K     18-Jun-82    MARC0037
CC2/CCC       Full   256     46     12K     22-Sep-82    MARC0037
CC3/CCC       Full   256     27      7K     22-Sep-82    MARC0037
CC4/CCC       Full   256     32      8K     21-Sep-82    MARC0037
CC6/CCC       Full   256     18      5K     31-Aug-82    MARC0037
CHGDATE/BAS   Full   256      4      1K     20-Oct-81    MARC0037
DABS/ASM      Full   256      2      1K     27-Sep-82    MARC0037
DADD/ASM      Full   256      3      1K     27-Sep-82    MARC0037

[listing continues]
```

<L>ist disks on file

The <L>ist function will display all the disks which are currently in the directory list. Press <L> from the master menu and the directory filename will be displayed at the top of the screen. The filename will be followed by a list of all the disk names on file with the free space available on that disk. ZCAT will pause after displaying a screen full of disk names. Press <ENTER> to continue the display or press <BREAK> to return to the master menu. The following illustrates such a listing:

```
         These disks are in catalog file : MARC/CAT:0

  Disk    Free    Disk    Free    Disk    Free    Disk    Free
========  ====  ========  ====  ========  ====  ========  ====
MARC0025    3K  MARC0026    3K  MARC0027    0K  MARC0028    0K
MARC0029    0K  MARC0030   33K  MARC0031    2K  MARC0032   15K
MARC0033    3K  MARC0034    9K  MARC0035   11K  MARC0036   32K
MARC0037    0K  MARC0038   84K  MARC0039   54K
```

<P>rint files in list

The <P>rint function will allow you to produce a hardcopy of your directory list. Press <P> from the master menu to obtain the print menu.

```
        *> P R I N T <*

    <F>iles by disk order
    <E>xpanded file order
    <C>ompressed file order

          Selection ?
```

Select the type of printout you would like by pressing the first character of the name or press <BREAK> to return to the master menu. You will be prompted:

```
Press <ENTER> when paper is set to top of form
```

When the paper has been positioned such that printing will begin on the first line of the paper, press <ENTER> to begin printing. Printing may be aborted at any time by pressing <BREAK>.

<F>iles by disk order

The disk names will be printed in alphabetical order. Printed with each disk name will be the amount of free space currently available on each disk and an alphabetical list of all files on that disk.

<E>xpanded file order

Files will be printed alphabetically, one across and 50 per page. The information printed for each file is the same as the "Display Directory" screen listing.

<C>ompressed file order

Files will be printed in alphabetical order, two per line and 100 per page. The following information will be printed for each file:

Filename	the name of the file
Mod date	the date the file was last written to
Disk name	the disk name where the file is located

<E>xit to GETSPEC

Press <E> to return to the GETSPEC menu. If changes have been made to the catalog file list, you will be prompted:

```
        Save changes ?
```

Press <Y> to save the changes or press <N> to return to GETSPEC without saving the changes. At the GETSPEC menu you may press <BREAK> to return to DOS Ready or you may select another catalog file.

Error messages

```
* * Disk is ALREADY cataloged * *
```

This error will occur when the disk name of disk which is being <A>dded is already in the catalog directory list. Use the <U>pdate function if the disk has already been cataloged. **Warning**: this error will also occur if two disks have the same disk name. If such is the case, change the disk name using the <C>hange function and then <A>dd it.

```
* * Disk is NOT cataloged * *
```

This error will occur when the disk name of a disk which is being <U>pdated is not in the catalog directory list. Use the <A>dd function to add a disk to the catalog list.

```
* * Disk name NOT found * *
```

This error will occur anytime ZCAT cannot locate a disk name in the catalog directory list. Check your spelling of the disk name.

```
* * Maximum file limit reached * *
```

This error will occur when the number of files in the catalog directory list becomes equal to the maximum number of files ZCAT can currently hold.

```
* * Maximum disk limit reached * *
```

This error will occur when 255 disks have been cataloged in one catalog directory list.

```
* * File too large for available memory * *
```

This error will occur when the requested directory file contains more entries than the current free memory will allow ZCAT to hold. Return to DOS Ready and free up some high memory [you may have to reboot and alter your high memory configuration].

DOS error messages

If an error occurs during disk I/O the standard DOS error message will be displayed. Press <ENTER> to return to the master menu.

Glossary of Terms

ABORT Terminate an operation, usually with the <BREAK> key

ADDRESS The location in memory of a particular byte, word, or string.

ALPHABETIC Any of the ASCII lower case characters "a" through "z" or upper case "A" through "Z".

ALPHABETICAL Pertains to sorting a list of items in an order which relates the items as alphabetic characters.

ALPHANUMERICS Any of the ASCII characters including "alphabetic" and numeric, "0" through "9".

ARROW A general term used to specify any of the four keyboard arrow keys: <LEFT>, <RIGHT>. <UP>, and <DOWN>.

ASCENDING A sorted order indicating an arrangement of items from lowest value to highest value. Sorted alphabetically, the arrangement would proceed from "AAAAA" through "Z

ASCII An acronym for the American Standard Code for Information Interchange. This standard describes the character set ranging from a value of 0 (decimal) through 127 (decimal).

ATTRIBUTE A facet of a file indicative of such things as whether the file is invisible to the DIRectory command, whether it's a SYSTEM file, whether it's a Partitioned Data Set, etc.

AUTO A command provided in your operating system to schedule the automatic operation of a command line when your computer is started.

BACKSPACE The keystroke used to back up and erase the character you just typed. A standard Model 4 keyboard uses the <LEFT ARROW> key for backspace; model 4D computers have a <BKSP> key which duplicates the <LEFT ARROW>.

BACKUP
An operating system command used to make a duplicate copy of a diskette or group of files.

BANK
A term which refers to a portion of the extended memory of your computer. A "bank" represents 32,768 (32K) characters of storage.

BAS
A file extension used to designate a program written in the BASIC language.

BASIC
The computer language supplied with your disk operating system.

BINARY
A number system consisting entirely of 0's and 1's an environment which can exist in either of two states (usually ON or OFF).

BIT
The smallest unit contained in a byte which is composed of eight bits.

BOOT
The action of starting up your computer by turning on its power, inserting a system disk into the floppy drive numbered 0, closing the drive door, and possibly depressing the RESET switch.

BREAK
A key labeled as such on your keyboard; this key is sometimes used by applications to abort a command operation.

BUFFER
A specific memory space reserved usually for intermediate storage of data being written to or read from a disk sector.

BUG
A malfunction of a computer program caused by an error in the program code. We hope there are none of these critters in this package.

Glossary of Terms

BYTE

The smallest storage unit in your computer. A byte can contain a number value of from zero through 255; a character is usually represented by a byte value. A byte contains eight bits.

CATALOG

A collection of information on files indicating what disk they are stored on and other relevant data such as file size, modification date, etc.

CHARACTER

A term applied to the symbols which make up the language we use to communicate with our computer and with which it communicates to us. These symbols may be alphanumerics, special characters (!@#$%^&*()-=+...), or non-printing control characters.

CIM

A file extension of files usually containing a Core Image Module. Such files are usually executable machine code in memory image form.

CLEAR

A key on your keyboard used to provide alternate functions when pressed in combination with another key or keys.

CMD

A file extension of executable command files. These are programs which can be run by typing the filename.

COM

A driver program provided with your operating system used to "connect" the DOS to your hardware serial port (RS-232C).

COMM

A utility program provided with your DOS to perform communications with another computer.

COMMAND

The general term applied to the many functions available in a program.

COMPARE

The action of checking two files or disks thought to be identical images of each other to ascertain if they are, indeed exact images.

CONCATENATE
The act of connecting together two or more records, files, or other similar entity.

CONFIGURATION
The current operating environment of your computer indicating such things as resident and active filters, condition of alterable DOS flags, device assignments, interrupt handlers, etc.

CONTROL
The term applied to a character value, usually in the range 0 through 31, which performs some special operation on a device or program.

CORE
An old term which indicated a particular type of memory storage based on magnetic cores. It is generally is accepted as a synonym of random access memory (RAM) accessible to a central processing unit (CPU).

CR
An abbreviation for "carriage return". This has an ASCII character value of 13 decimal; it is entered on your keyboard via the key labeled. <ENTER>.

CTRL
This is the mnemonic on the keyboard key used to enter control characters; it is used in combination with another key.

CURSOR
Anytime your computer is waiting for you to enter a keystroke, its position on the video screen is usually marked with a special character. This character may be an underline, a block, or a right angle bracket. This character is the "cursor".

CYLINDER
A term to designate all like numbered tracks on all surfaces of a multi-surface disk drive. A two-headed drive would have two surfaces and two tracks per cylinder. On a one-headed disk drive, a track and a cylinder are synonymous.

DAT
A file extension of files usually containing data.

DATA
Information usually stored in a disk file.

Glossary of Terms

DCB

An acronym for Device Control Block; it is a system memory storage space associated with character devices (video, keyboard, printer, serial port, etc.).

DDEN

An acronym for double density formatting associated with floppy diskettes.

DECIMAL

A number system based on the ten digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.

DEFAULT

A particular operation or configuration which is installed or designated without any specific action by you.

DELETE

The action of removing, killing, or erasing a file, a record, a field, or any similar entity.

DENSITY

A term applicable to the method of recording data onto a disk drive media. For floppy diskettes, the methods are usually single or double density.

DEPRESS

The action of tapping, momentarily pressing on, or otherwise engaging one of the keys on a keyboard.

DEVICE

A term applied to a peripheral component of your computer usually associated with character input/output (e.g. keyboard device, video device, printer device, etc.).

DIGIT

One of the characters of a numbering system. Binary digits are the two characters 0 and 1; octal digits are the eight characters 0, 1, 2, 3, 4, 5, 6, and 7; decimal digits are the ten characters 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9; and hexadecimal digits are the sixteen characters 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

DIR

An operating system library command used to display the names and other storage details of the files stored on a floppy or hard disk.

DIRECTORY

A special file on a floppy or hard disk which contains the location and other pertinent details of all files stored on that disk.

DISK

The abbreviated term for "disk drive" which is the hardware apparatus used to either store your files, in the case of a hard or rigid disk drive, or is used to read your floppy diskettes, in the case of a floppy disk drive.

DISKETTE

A shorthand way of saying "floppy diskette", the flexible recording medium used by your computer's disk drives to store files.

DISPLAY

A shorthand way of referring to the video display screen of your computer.

DNARW

A mnemonic designating the <DOWN ARROW> key on your keyboard.

DO

An operating system command used to initiate the commands stored in a Job Control Language file (JCL).

DOS

An acronym for Disk Operating System. This is the system supplied with your computer which manages the files stored on diskettes and provides an operating environment for programs.

DRIVE

An abbreviation for *disk drive".

DRIVER

A special computer program connecting a hardware device to the DOS (e.g. COM/DVR).

DRIVESPEC

The disk drive identifier field of the file specification. This is indicated by a colon followed by a number in the range 0-7.

Glossary of Terms

EOF

An abbreviation for "end of file". It is a position in a file indicating where the last byte is stored.

EXECUTE

An action which causes a computer program to start operating.

EXTENSION

One of the fields of a file specification generally used to indicate a particular class of file such as "/DAT" for data, "/CMD" for command, "/BAS" for BASIC, etc.

FIELD

One particular item in a group of data making up a record.

FILE

A collection of data records stored on disk.

FILENAME

The name field of a file specification.

FILESPEC

An abbreviation for file specification: the entire character string which identifies a particular file. It is composed of a file name, a file extension, a password, and a drivespec.

FILTER

A special program inserted logically into a device stream for the purpose of altering the behavior of input/output.

FLAG

A data field used to store a particular envi ronment state. Flags are usually binary (e.g. ON/OFF, YES/NO, TRUE/FALSE).

FLOATING POINT

A term applied to real numbers (in the mathematical sense). Floating point number will mostly always contain fractional values.

FLOPPY

An abbreviation for "floppy diskette"; it is the flexible recording medium used to store files.

FORM

A pre-determined layout of fields positioned on the video display screen or on paper.

FORMAT
An operating system utility used to prepare a blank diskette before files can be stored on it, or other diskettes backed up to it.

FULL
One of the types of access provided to files. Others are READ, WRITE, EXEC, etc.

HEADER
Generally used to indicate a specific record at tached to the front of a file which contains data used by the program or routine which reads the file.

HEXADECIMAL
A base-16 character value; hexadecimal digits are the sixteen characters 0, 1. 2. 3, 4. 5, 6, 7. 8, 9, A, B, C, D. E, and F.

HIT
An acronym for Hash Index Table. It is a portion of the directory stored on a disk which contains the one-byte directory entry codes for each active file.

HSB
An acronym for "high-order significant byte"; this is the highest storage portion of a three-byte value.

INSERT
The action of adding a character into a screen position by shifting all characters from the cursor position one place to the right.

INTEGER
A whole number, sometimes referred to as a counting number. Integer numbers have no fractional part.

INVERSE
Another term for "reverse" associated with the videodisplay screen. Normal displayed characters are white foreground on a black background. Inverse, or reverse, video characters are black foreground on a white background.

INVISIBLE
An attribute which can be placed on a file by the ATTRIB DOS library command which inhibits

the file's information from being presented by the DIR DOS library command.

INVOKE
The name applied to the function of gaining run access to an application.

ISAM
An acronym for "Indexed Sequential Access Method"; used for a particular type of file which contains a sequential set of records or members with an index to the beginning of each record or member.

JCL
An acronym for Job Control Language. This is a DOS facility to execute a predetermined set of command lines.

KEY
One of the keys on your keyboard; sometimes used to refer to the computer action of scanning the keyboard to see if a key is depressed.

KEYIN
The acronym for "keyboard line input"; a DOS function used to obtain a line of keystrokes.

KEYSTROKE
The result of depressing one or more keyboard keys to generate a particular character value.

KI
The name associated with the keyboard input device; designated as "*KI" when entering a device specification.

KSM
An acronym for KeyStroke Multiplication; a filter provided with your DOS which lets you generate a string of characters by depressing a single keystroke.

LANGUAGE
The words, structure, and symbols constituting a particular programming environment. Typical programming languages are ASSEMBLER, BASIC, C, FORTRAN, and PASCAL.

LIBRARY
A set of application modules stored together in one file and which contains an internal directory.

Also the general term of many DOS commands identified by issuing a LIB command.

LOAD
The act performed by a LOADER.

LOADER
A program or routine used to read another file, which usually contains a computer program, and store it in memory to be run.

LOGICAL
The term applied to an event which is simulated rather than physical. A character value of 127 decimal may be interpreted as a carriage return for export purposes; since 127 is not physically a 13 decimal, it is termed, in this case, a logical carriage return.

LOWERCASE
Any of the alphabetic characters "a" through "z" entered normally by not depressing the <SHIFT> key nor by engaging <CAPS>.

LRL
An acronym for Logical Record Length; it represents the length of a record in bytes. Since the record length is not a physical phenomena, it is termed logical.

LSB
An acronym for "low-order significant byte"; this is the lowest storage portion of a three-byte value.

LS-DOS
A nomenclature of the 6.3 release of the operating system used on the Model 4 computer.

MACRO
The term applied to string of characters automatically generated from a single character. This is similar in theory to KSM.

MAINTENANCE
Repetitive actions performed at periodic intervals to keep something working, or to repair something that is broken.

MASK	A character or characters overlaying another character or characters in a special manner as to eliminate certain bit positions of the result.
MEDIA	A term indicating the magnetic or optical surface of material used to store computer files.
MEMBER	One of the applications stored in a library.
MEMBERSPEC	The specification of a library member which denotes its file name, its application name, and the disk drive on which the library file is stored.
MEMORY	The storage area for characters and programs internal to your machine (in contrast to disk storage).
MENU	The list of commands supported by an application which are displayed as a screen or a portion of a screen.
MESSAGE	A phrase or sentence used to inform you of some event. It may apprise you of an error or prompt you for a particular response.
MNEMONIC	An abbreviation or label for an entity.
MODEM	A hardware peripheral device used to communicate with another computer over a telephone or data line.
MODULE	Another name meaning "member".
MSB	An acronym for "mid-order significant byte"; this is the middle storage portion of a three-byte value.
MSPEC	An abbreviation for "memberspec".
NREC	An acronym for "number of records".

OCTAL
A base 8 numbering system; octal digits are the eight characters 0, 1, 2, 3, 4, 5, 6, and 7.

OFFSET
A position of a field, byte, or other piece of data which is relative to the beginning of a record.

ORIGIN
The memory address used to store the first byte of a program.

OVERLAY
A region of the DOS used for swapping in different processes.

OVERSTRIKE
The action of typing one character over another; the second replacing the first.

PaDS
An acronym for Partitioned Data Set. It is a name applied to the files managed by the utility of the same name; these files are similar to the DOS library files.

PARAMETER
An option, usually entered on the command line, which alters the behavior of a program.

PARTITION
Any one piece of a unit, commonly a hard disk drive, which has been divided up into more than one logical unit.

PASSWORD
A string of characters required as part of a file specification or entire disk before a given level of access is permitted.

PATCH
A DOS utility which applies alterations to disk files. The term also applies to the modifications.

PDS
Another "older" way of writing "PaDS".

PORT
A physical interface of your computer used to connect a peripheral piece of equipment (e.g. printer port, RS-232 serial port).

PRECISION
The amount of significance contained in a floating point number. The best illustration is to

conceptualize a bulls-eye target's shot pattern. A tight cluster of shots some distance from the bulls eye would have a high degree of precision but little accuracy.

PROGRAM A predetermined sequence of machine instructions which together support some reasonably complex operations.

PROMPT A displayed message which expects a usable response.

PROTECTION Measures applied to a file for security purposes. Itmay relate to access level restrictions or password control before granting access. Protection is applied via the DOS ATTRIB command.

PRO-WAM The trademarked name applied to the window controller and application manager published by MISOSYS.

PURGE The act of erasing, deleting, removing, or killing a group of files.

RAM A type of memory storage which provides both read and write capabilities.

README/TXT A plain text file included on your program diskette which contains last minute information not printed in this manual.

RECORD A collection of data fields associated with a particular key field or fields.

REGISTER A very fast memory storage location located within the Central Processing Unit (CPU) of your computer.

REMOVE The act of erasing, killing, or deleting a file, a module, a record, or other similar entity.

RENAME
The act of changing the filename and/or file extension of a file from one name to another.

RESIDENT
The term applied to a program which stays in the memory region of your computer after control is passed back to the DOS. The memory resident module usually embellishes, and is an extension to, the DOS.

RESPONSE
The entry you input into a program after viewing a particular message prompt.

RESTORE
The act of recovering a file archived to some place other than its working environment.

RETURN
Short for "carriage return"; see "CR".

REVERSE
Another term for "inverse" associated with the video display screen. Normal displayed characters are white foreground on a black background. Reverse, or inverse, video characters are black foreground on a white background.

RPN
An acronym for Reverse Polish Notation, commonly referred to as "infix". It relates to the syntax of entering numbers and operators for mathematical calculations.

RUN
The action of initiating program execution.

SCIENTIFIC
A format associated with the presentation of floating point numbers in fraction and exponent form.

SCREEN
Short for the video display screen.

SCROLL
The action of shifting the entire text display of the screen or a window of the screen either vertically upward or downward.

SEARCH The action of looking throughout a set of data records to find one which matches a key string.

SECTOR A physical storage unit of a floppy diskette of hard disk drive.

SERIAL A type of device where the character bytes are passed bit-wise; this contrasts with parallel where character bytes are passed as a full byte.

SETCOM A library command provided with your DOS used to alter the parameters associated with the serial driver (COM/DVR).

SORT The operation of placing a list of items into some designated order.

SPEC Short for "file specification".

STRING A series of character values, usually denoted on paper by enclosing in quotation marks (e.g. "this is a string").

SVC An acronym for "SuperVisor Call"; a facility of the DOS used to communicate with it at the program level in contrast to the command level.

SYNTAX A particular requirement for the entering of information.

SYS A particular file "tension indicating a "system" file; e.g. SYS0/SYS.

SYSGEN A command of the DOS used to store the current operating environment to the CONFIG/SYS file.

SYSTEM The nomenclature of a diskette which is used to either BOOT your computer or which is used in drive :0.

TED
A text editor application included with the Mister ED application pack; also bundled with LS-DOS 6.3 as a stand alone command.

TEMPLATE
An outline or "form" used to organize a set of data.

TOGGLE
Switch a binary device from its current state to its opposite state.

TRACK
A circular virtual groove of a disk surface; it contains the sectors read during one rotation of the disk.

TXT
A file extension used to designate a plain text file.

UPARW
A mnemonic for the <UP ARROW> key.

USER
Someone who operates computer programs but does no or little computer programming.

UTILITY
A computer program designed to do some maintenance operation.

VERIFY
The action of reading a sector after it is written to ensure that it remains readable. It provides an extra measure of insurance against media wear at the cost of additional time for disk I/O.

VIDEO
Short for the video display screen.

WILDCARD
A string of characters which don't completely designate a file specification but which contain certain "global" characters such as "*", "$", or "?" used to match any character thereby specifying a group of "matching" file specifications.

WORD
In the mathematical sense, the term applied to a 16-bit value which will occupy a two-byte storage region. Also is the name of the word processor used to prepare this documentation.